

LIS Source Code Documentation

Submitted under Task Agreement GSFC-CT-2

Cooperative Agreement Notice (CAN) CAN-00OES-01
Increasing Interoperability and Performance of

Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences

December 2003

Version 2.3

History:

Revision	Summary of Changes	Date
2.3	LIS 2.3 code release	December, 2003



NASA Goddard Space Flight Center,
Greenbelt, MD 20771

Contents

1 Routine/Function Prologues	8
1.0.1 absoft_release_cache.F90 (Source File: absoft.F90)	8
1.1 Fortran: Module Interface baseforcing_module.F90 (Source File: baseforcing_module.F90)	8
1.1.1 LIS_get_base_forcing (Source File: baseforcing_module.F90)	8
1.1.2 LIS_baseforcing_init (Source File: baseforcing_module.F90)	8
1.1.3 forcing_init (Source File: baseforcing_module.F90)	9
1.1.4 init_baseforcing (Source File: baseforcing_module.F90)	9
1.1.5 get (Source File: baseforcing_module.F90)	10
1.1.6 time_interp (Source File: baseforcing_module.F90)	10
1.1.7 ld_getbaseforcing (Source File: baseforcing_module.F90)	11
1.1.8 scatter_data (Source File: baseforcing_module.F90)	11
1.1.9 define_gds.F90 (Source File: define_gds.F90)	12
1.2 Fortran: Module Interface def_ipMod.F90 (Source File: def_ipMod.F90) . . .	15
1.2.1 allocate_ip (Source File: def_ipMod.F90)	15
1.2.2 def_ip_input (Source File: def_ipMod.F90)	16
1.3 Fortran: Module Interface driverpardef_module.F90 (Source File: driverpardef_module.F90)	18
1.3.1 def_driverpar_structs (Source File: driverpardef_module.F90)	18
1.3.2 elevadjust.F90 (Source File: elevadjust.F90)	19
1.3.3 endrun.F90 (Source File: endrun.F90)	20
1.4 Fortran: Module Interface grid_module.F90 (Source File: grid_module.F90)	21
1.5 Fortran: Module Interface grid_spmdMod.F90 (Source File: grid_spmdMod.F90)	22
1.5.1 allocate_gdd (Source File: grid_spmdMod.F90)	22
1.5.2 grid_spmd_init (Source File: grid_spmdMod.F90)	23
1.5.3 lisdrv.F90 - Main program for LIS (Source File: lisdrv.F90)	23
1.6 Function calls for main routines:	23
1.7 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90)	25
1.7.1 ld_domain_init (Source File: lisdrv_module.F90)	25
1.7.2 setup_timeMgr (Source File: lisdrv_module.F90)	26
1.7.3 LIS_ticktime (Source File: lisdrv_module.F90)	26
1.7.4 LIS_allocate_memory (Source File: lisdrv_module.F90)	27
1.7.5 setnch (Source File: lisdrv_module.F90)	29
1.7.6 getdomain (Source File: lisdrv_module.F90)	29
1.7.7 getlsm (Source File: lisdrv_module.F90)	30
1.7.8 getnch (Source File: lisdrv_module.F90)	30
1.7.9 getnc (Source File: lisdrv_module.F90)	30
1.7.10 getnr (Source File: lisdrv_module.F90)	31
1.7.11 getmaxt (Source File: lisdrv_module.F90)	31
1.7.12 getforcing (Source File: lisdrv_module.F90)	31
1.7.13 getnmif (Source File: lisdrv_module.F90)	32
1.7.14 LIS_endofrun (Source File: lisdrv_module.F90)	32
1.7.15 dist_gindex (Source File: lisdrv_module.F90)	32
1.8 Fortran: Module Interface lis_module.F90 (Source File: lis_module.F90) . .	34
1.9 Fortran: Module Interface lsm_module.F90 (Source File: lsm_module.F90) . .	37
1.9.1 LIS_lsm_init (Source File: lsm_module.F90)	37

1.9.2	LIS_allocate_lsm (Source File: lsm_module.F90)	38
1.9.3	LIS_setuplsm (Source File: lsm_module.F90)	38
1.9.4	LIS_lsm_main (Source File: lsm_module.F90)	38
1.9.5	LIS_force2tile (Source File: lsm_module.F90)	38
1.9.6	LIS_readrestart (Source File: lsm_module.F90)	38
1.9.7	LIS_writerestart (Source File: lsm_module.F90)	38
1.9.8	LIS_lsm_output (Source File: lsm_module.F90)	39
1.9.9	LIS_setDynlsm (Source File: lsm_module.F90)	39
1.9.10	lsm_tile_allocate (Source File: lsm_module.F90)	39
1.9.11	lsm_setup (Source File: lsm_module.F90)	40
1.9.12	run_lsm (Source File: lsm_module.F90)	40
1.9.13	lsm_readrestart (Source File: lsm_module.F90)	41
1.9.14	write_output (Source File: lsm_module.F90)	41
1.9.15	setDynParams (Source File: lsm_module.F90)	41
1.9.16	lsm_f2t (Source File: lsm_module.F90)	42
1.9.17	lsm_writerestart (Source File: lsm_module.F90)	42
1.9.18	maketiles_gds() (Source File: maketiles_gds.F90)	43
1.9.19	maketiles_nongds.F90 (Source File: maketiles_nongds.F90)	52
1.10	Fortran: Module Interface mpishorthand.F90 (Source File: mpishorthand.F90)	61
1.11	Fortran: Module Interface obsprecipforcing_module.F90 (Source File: obsprecipforcing_module.F90)	61
1.11.1	LIS_obsprecipforcing_init (Source File: obsprecipforcing_module.F90)	62
1.11.2	LIS_get_obsprecip_forcing (Source File: obsprecipforcing_module.F90)	62
1.11.3	precip_forcing_init (Source File: obsprecipforcing_module.F90) . . .	62
1.11.4	get_precip_forcing (Source File: obsprecipforcing_module.F90) . . .	63
1.11.5	scatter_precip_data (Source File: obsprecipforcing_module.F90) . . .	63
1.12	Fortran: Module Interface obsradforcing_module.F90 (Source File: obsradforcing_module.F90)	63
1.12.1	LIS_obsradforcing_init (Source File: obsradforcing_module.F90) . . .	64
1.12.2	LIS_get_obsrad_forcing (Source File: obsradforcing_module.F90) . . .	64
1.12.3	rad_forcing_init (Source File: obsradforcing_module.F90)	64
1.12.4	get_obsrad_forcing (Source File: obsradforcing_module.F90)	65
1.12.5	scatter_rad_data (Source File: obsradforcing_module.F90)	66
1.13	Fortran: Module Interface opendap_module.F90 (Source File: opendap_module.F90)	66
1.13.1	opendap_init (Source File: opendap_module.F90)	67
1.13.2	reset_lis_filepaths (Source File: opendap_module.F90)	68
1.13.3	init_parm_vars (Source File: opendap_module.F90)	68
1.13.4	set_parm_lat (Source File: opendap_module.F90)	69
1.14	Fortran: Module Interface precision.F90 (Source File: precision.F90) . . .	70
1.14.1	readcard.F90 (Source File: readcard.F90)	71
1.14.2	check_timestep (Source File: readcard.F90)	73
1.14.3	openfile (Source File: readcard.F90)	74
1.15	Fortran: Module Interface spmdMod.F90 (Source File: spmdMod.F90) . . .	75
1.15.1	spmd_init (Source File: spmdMod.F90)	76
1.15.2	stats.F90 (Source File: stats.F90)	76
1.16	Fortran: Module Interface string_utils (Source File: string_utils.F90) . . .	77
1.16.1	to_upper (Source File: string_utils.F90)	77
1.16.2	t2gr.F90 (Source File: t2gr.F90)	78

1.16.3 tile2grid.F90 (Source File: tile2grid.F90)	78
1.16.4 tile_module.F90 (Source File: tile_module.F90)	79
1.17 Fortran: Module Interface tile_spmdMod.F90 (Source File: tile_spmdMod.F90)	80
1.17.1 allocate_tiledd (Source File: tile_spmdMod.F90)	80
1.17.2 tile_spmd_init (Source File: tile_spmdMod.F90)	80
1.18 Fortran: Module Interface time_manager.F90 (Source File: time_manager.F90)	81
1.18.1 timemgr_init (Source File: time_manager.F90)	83
1.18.2 timemgr_print (Source File: time_manager.F90)	85
1.18.3 timemgr_print (Source File: time_manager.F90)	85
1.18.4 advance_timestep (Source File: time_manager.F90)	86
1.18.5 get_step_size (Source File: time_manager.F90)	86
1.18.6 get_nstep (Source File: time_manager.F90)	87
1.18.7 get_curr_day (Source File: time_manager.F90)	87
1.18.8 get_prev_date (Source File: time_manager.F90)	88
1.18.9 get_start_date (Source File: time_manager.F90)	88
1.18.10 get_ref_date (Source File: time_manager.F90)	88
1.18.11 get_curr_time (Source File: time_manager.F90)	89
1.18.12 get_curr_calday (Source File: time_manager.F90)	89
1.18.13 is_end_curr_day (Source File: time_manager.F90)	90
1.18.14 is_end_curr_month (Source File: time_manager.F90)	90
1.18.15 is_first_step (Source File: time_manager.F90)	91
1.18.16 is_first_restart_step (Source File: time_manager.F90)	91
1.18.17 is_last_step (Source File: time_manager.F90)	91
1.18.18 timemgr_write_restart (Source File: time_manager.F90)	91
1.18.19 timemgr_read_restart (Source File: time_manager.F90)	92
1.18.20 chkrc (Source File: time_manager.F90)	92
1.19 Fortran: Module Interface time_module.F90 (Source File: time_module.F90)	93
1.19.1 zterp.F90 (Source File: zterp.F90)	98
1.19.2 coszenith (Source File: zterp.F90)	103
1.19.3 locltime (Source File: zterp.F90)	105
1.20 Fortran: Module Interface geosdomain_module.F90 (Source File: geosdomain_module.F90)	106
1.20.1 defnatgeos.F90 (Source File: geosdomain_module.F90)	106
1.21 Fortran: Module Interface geosdrv_module.F90 (Source File: geosdrv_module.F90)	107
1.22 Fortran: Module Interface geosopendap_module.F90 (Source File: geosopendap_module.F90)	107
1.22.1 reset_geos_filepaths (Source File: geosopendap_module.F90)	108
1.22.2 init_geos_vars (Source File: geosopendap_module.F90)	109
1.22.3 def_kgds (Source File: geosopendap_module.F90)	110
1.22.4 init_geos_ref_date (Source File: geosopendap_module.F90)	111
1.22.5 get_geos_index (Source File: geosopendap_module.F90)	111
1.22.6 set_geos_lat (Source File: geosopendap_module.F90)	112
1.22.7 getgeos.F90 (Source File: getgeos.F90)	113
1.23 Core Functions of getgeos	113
1.23.1 geosfile (Source File: getgeos.F90)	117
1.23.2 readgeoscrd.F90 (Source File: readgeoscrd.F90)	119
1.23.3 readgeos.F90 (Source File: readgeos.F90)	120
1.24 Core Functions of readgeos	120

1.24.1	check_error (Source File: readgeos.F90)	126
1.24.2	time_interp_geos.F90 (Source File: time_interp_geos.F90)	127
1.25	Core Functions of time_interp_geos	127
1.26	Fortran: Module Interface gdasdomain_module.F90 (Source File: gdasdomain_module.F90)	130
1.26.1	defnatgdas.F90 (Source File: gdasdomain_module.F90)	130
1.27	Fortran: Module Interface gdasdrv_module.F90 (Source File: gdasdrv_module.F90)	131
1.27.1	getgdas.F90 (Source File: getgdas.F90)	131
1.28	Core Functions of getgdas	132
1.28.1	gdasfile (Source File: getgdas.F90)	137
1.28.2	gdasfilef06 (Source File: getgdas.F90)	139
1.28.3	readgdascrd.F90 (Source File: readgdascrd.F90)	141
1.28.4	retgdas.F90 (Source File: retgdas.F90)	142
1.28.5	interp_gdas (Source File: retgdas.F90)	144
1.28.6	time_interp_gdas (Source File: time_interp_gdas.F90)	146
1.29	Core Functions of time_interp_gdas	146
1.30	Fortran: Module Interface baseforcing_pluginMod.F90 (Source File: baseforcing_pluginMod.F90)	149
1.30.1	baseforcing_plugin (Source File: baseforcing_pluginMod.F90)	149
1.31	Fortran: Module Interface precipforcing_pluginMod.F90 (Source File: precipforcing_pluginMod.F90)	150
1.31.1	precipforcing_plugin (Source File: precipforcing_pluginMod.F90)	150
1.32	Fortran: Module Interface radforcing_pluginMod.F90 (Source File: radforcing_pluginMod.F90)	151
1.32.1	radforcing_plugin (Source File: radforcing_pluginMod.F90)	151
1.33	Fortran: Module Interface lsm_pluginMod.F90 (Source File: lsm_pluginMod.F90)	152
1.33.1	lsm_plugin (Source File: lsm_pluginMod.F90)	152
1.33.2	clm2_almaout.F90 (Source File: clm2_almaout.F90)	155
1.34	Fortran: Module Interface clm2drv_module.F90 (Source File: clm2drv_module.F90)	157
1.34.1	clm2_dynsetup.F90 (Source File: clm2_dynsetup.F90)	158
1.34.2	clm2_gather (Source File: clm2_gather.F90)	159
1.34.3	clm2_gridout.F90 (Source File: clm2_gridout.F90)	159
1.34.4	clmlairead.F90: (Source File: clm2lairead.F90)	169
1.34.5	avhrr_file_2 (Source File: clm2lairead.F90)	178
1.34.6	avhrr_file_5KM (Source File: clm2lairead.F90)	180
1.34.7	ncarlai_clm2 (Source File: clm2lairead.F90)	182
1.34.8	ncarcanht_clm2 (Source File: clm2lairead.F90)	184
1.34.9	modis_file_2 (Source File: clm2lairead.F90)	186
1.34.10	clm2_output.F90 (Source File: clm2_output.F90)	188
1.35	Fortran: Module Interface clm2pardef_module.F90 (Source File: clm2pardef_module.F90)	189
1.35.1	def_clmpar_struct (Source File: clm2pardef_module.F90)	190
1.35.2	clm2_restart.F90 (Source File: clm2_restart.F90)	190
1.35.3	clm2_scatter.F90 (Source File: clm2_scatter.F90)	191
1.35.4	clm2_setup.F90 (Source File: clm2_setup.F90)	191
1.35.5	clm2_singlegather.F90 (Source File: clm2_singlegather.F90)	193
1.35.6	clm2_singleout.F90 (Source File: clm2_singleout.F90)	196
1.35.7	clm2_tileout.F90 (Source File: clm2_tileout.F90)	199
1.35.8	clm2_totinit.F90 (Source File: clm2_totinit.F90)	209

1.35.9	clm_varder.F90 (Source File: clm_varder.F90)	210
1.35.10	clm_varder.ini (Source File: clm_varder.F90)	211
1.35.11	clm_varder_init (Source File: clm_varder.F90)	212
1.35.12	iniTimeConst.F90 (Source File: iniTimeConst.F90)	212
1.35.13	iniTimeVar.F90 (Source File: iniTimeVar.F90)	219
1.35.14	mapvegc.F90 (Source File: mapvegc.F90)	225
1.35.15	noah_alb.F90: (Source File: noah_alb.F90)	226
1.35.16	noah_almaout.F90 (Source File: noah_almaout.F90)	230
1.35.17	noah_atmdrv.F90 (Source File: noah_atmdrv.F90)	233
1.35.18	noah_coldstart.F90 (Source File: noah_coldstart.F90)	233
1.36	Fortran: Module Interface noahdrv_module.F90 (Source File: noahdrv_module.F90)	234
1.36.1	noah_dynsetup.F90 (Source File: noah_dynsetup.F90)	235
1.36.2	noah_gather.F90 (Source File: noah_gather.F90)	236
1.36.3	noah_gfrac.F90 (Source File: noah_gfrac.F90)	237
1.36.4	noah_gridout.F90 (Source File: noah_gridout.F90)	240
1.36.5	noah_main.F90 (Source File: noah_main.F90)	247
1.37	Fortran: Module Interface noah_module.F90 (Source File: noah_module.F90)	261
1.37.1	noah_output.F90 (Source File: noah_output.F90)	262
1.38	Fortran: Module Interface noahpardef_module.F90 (Source File: noahpardef_module.F90)	263
1.38.1	def_noahpar_struct (Source File: noahpardef_module.F90)	264
1.38.2	noahrst.F90 (Source File: noahrst.F90)	295
1.38.3	noah_scatter.F90 (Source File: noah_scatter.F90)	297
1.38.4	noah_setup.F90 (Source File: noah_setup.F90)	297
1.38.5	noah_singlegather.F90 (Source File: noah_singlegather.F90)	298
1.38.6	noah_singleout.F90 (Source File: noah_singleout.F90)	301
1.38.7	noah_soiltype.F90 (Source File: noah_soil_typ.F90)	304
1.38.8	noah_tileout.F90 (Source File: noah_tileout.F90)	306
1.38.9	noah_totinit.F90 (Source File: noah_totinit.F90)	312
1.38.10	noah_varder.F90 (Source File: noah_varder.F90)	313
1.38.11	noah_varder_ini (Source File: noah_varder.F90)	314
1.38.12	noah_writerst.F90 (Source File: noah_writerst.F90)	315
1.38.13	readnoahcrd.F90 (Source File: readnoahcrd.F90)	318
1.38.14	setnoahp.F90 (Source File: setnoahp.F90)	319
1.38.15	sh2o_init.F90 (Source File: sh2o_init.F90)	324
1.38.16	opendap_init_c_struct (Source File: opendap_wrappers.F90)	325
1.38.17	readviccrd.F90 (Source File: readviccrd.F90)	326
1.38.18	vic_atmdrv.F90 (Source File: vic_atmdrv.F90)	327
1.39	Fortran: Module Interface vicdrv_module.F90 (Source File: vicdrv_module.F90)	327
1.39.1	vic_dynsetup.F90 (Source File: vic_dynsetup.F90)	328
1.39.2	vic_main.F90 (Source File: vic_main.F90)	328
1.39.3	vic_output.F90 (Source File: vic_output.F90)	329
1.40	Fortran: Module Interface vicpardef_module.F90 (Source File: vicpardef_module.F90)	330
1.40.1	def_vicpar_struct (Source File: vicpardef_module.F90)	330
1.40.2	vic_readrestart.F90 (Source File: vic_readrestart.F90)	330
1.40.3	vic_setup.F90 (Source File: vic_setup.F90)	331
1.41	Fortran: Module Interface vic_varder.F90 (Source File: vic_varder.F90) . . .	332
1.41.1	vic_varder_ini (Source File: vic_varder.F90)	333
1.41.2	vic_writerst.F90 (Source File: vic_writerestart.F90)	333

1.42 Fortran: Module Interface cmapdomain_module.F90 (Source File: cmapdomain_module.F90)	334
1.42.1 defnatcmap.F90 (Source File: cmapdomain_module.F90)	335
1.42.2 allocate_cmap_ip (Source File: cmapdomain_module.F90)	335
1.42.3 def_cmap_ip_input (Source File: cmapdomain_module.F90)	336
1.43 Fortran: Module Interface cmapdrv_module.F90 (Source File: cmapdrv_module.F90)	337
1.43.1 getcmap.F90 (Source File: getcmap.F90)	338
1.43.2 cmapfile (Source File: getcmap.F90)	339
1.43.3 glbprecip_cmap.F90 (Source File: glbprecip_cmap.F90)	341
1.43.4 interp_cmap.F90 (Source File: interp_cmap.F90)	342
1.43.5 readcmapcrd.F90 (Source File: readcmapcrd.F90)	344
1.43.6 time_interp_cmap.F90 (Source File: time_interp_cmap.F90)	345
1.43.7 gethuff.F90 (Source File: gethuff.F90)	346
1.43.8 hufffile (Source File: gethuff.F90)	349
1.43.9 glbprecip_huff.F90 (Source File: glbprecip_huff.F90)	350
1.44 Fortran: Module Interface huffdomain_module.F90 (Source File: huffdomain_module.F90)	352
1.44.1 defnathuff.F90 (Source File: huffdomain_module.F90)	352
1.44.2 allocate_huff_ip (Source File: huffdomain_module.F90)	353
1.45 Fortran: Module Interface huffdrv_module.F90 (Source File: huffdrv_module.F90)	353
1.45.1 readhuffcrd.F90 (Source File: readhuffcard.F90)	354
1.45.2 time_interp_huff.F90 (Source File: time_interp_huff.F90)	354
1.45.3 getpers.F90 (Source File: getpers.F90)	354
1.45.4 persfile (Source File: getpers.F90)	356
1.45.5 glbprecip_pers.F90 (Source File: glbprecip_pers.F90)	357
1.46 Fortran: Module Interface persdomain_module.F90 (Source File: persdomain_module.F90)	358
1.46.1 defnatpers.F90 (Source File: persdomain_module.F90)	359
1.46.2 allocate_pers_ip (Source File: persdomain_module.F90)	360
1.47 Fortran: Module Interface persdrv_module.F90 (Source File: persdrv_module.F90)	360
1.47.1 readperscrd.F90 (Source File: readperscard.F90)	361
1.47.2 time_interp_pers.F90 (Source File: time_interp_pers.F90)	361
1.47.3 agrlwdn.F90 (Source File: agrlwdn.F90)	361
1.48 Fortran: Module Interface agrmetdomain_module.F90 (Source File: agrmetdomain_module.F90)	363
1.48.1 defnatagrmet.F90 (Source File: agrmetdomain_module.F90)	364
1.48.2 allocate_agr_ip (Source File: agrmetdomain_module.F90)	365
1.48.3 def_agr_ip_input (Source File: agrmetdomain_module.F90)	365
1.49 Fortran: Module Interface agrmetdrv_module.F90 (Source File: agrmetdrv_module.F90)	366
1.49.1 getgrad.F90 (Source File: getgrad.F90)	367
1.49.2 agrSWfile: (Source File: getgrad.F90)	370
1.49.3 subroutine interp_agrmet_lw.F90 (Source File: interp_agrmet_lw.F90)	371
1.49.4 interp_agrmet_sw.F90 (Source File: interp_agrmet_sw.F90)	373
1.49.5 readagrmetcrd.F90 (Source File: readagrmetcrd.F90)	375
1.49.6 retagrlw.F90 (Source File: retagrlw.F90)	375
1.49.7 retglbSW.F90 (Source File: retglbSW.F90)	378
1.50 Fortran: Module Interface time_interp_agrmet.F90 (Source File: time_interp_agrmet.F90)	379

1 Routine/Function Prologues

1.0.1 absoft_release_cache.F90 (Source File: absoft.F90)

This routine forces the explicit clearing of memory caches when using the absoft compiler.

REVISION HISTORY:

10 Sept 03 Jim Geiger Initial Specification

INTERFACE:

```
subroutine absoft_release_cache()
```

1.1 Fortran: Module Interface baseforcing_module.F90 (Source File: baseforcing_module.F90)

This module contains interfaces and subroutines that controls the incorporation of model forcing

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

1.1.1 LIS_get_base_forcing (Source File: baseforcing_module.F90)

INTERFACE:

```
interface LIS_get_base_forcing
    module procedure ld_getbaseforcing
end interface
```

1.1.2 LIS_baseforcing_init (Source File: baseforcing_module.F90)

INTERFACE:

```
interface LIS_baseforcing_init
    module procedure init_baseforcing
end interface
```

1.1.3 forcing_init (Source File: baseforcing_module.F90)

Sets up functions for reading model forcing

INTERFACE:

```
subroutine forcing_init()
```

USES:

```
use baseforcing_pluginMod
```

1.1.4 init_baseforcing (Source File: baseforcing_module.F90)

Initializes model forcing variables and allocates memory

INTERFACE:

```
subroutine init_baseforcing()
```

USES:

```
use def_ipMod, only: def_ip_input, allocate_ip, &
    def_budgetip_input
use lisdrv_module, only: lis, getforcing
#ifndef OPEN DAP
    use opendap_module
#endif
integer :: kgdsi(200)
```

CONTENTS:

```
kgdsi = 0
call forcing_init()
call allocate_forcing_mem()
#ifndef OPEN DAP
    call defnates(getforcing(),kgdsi)
    call define_gds(lis)
    call allocate_ip(parm_nc*parm_nr)
    call def_ip_input(kgdsi)
#else
    if( masterproc) then
        call defnates(getforcing(),kgdsi)
        call allocate_ip(lis%d%lnc*lis%d%lnr)
        call def_ip_input(kgdsi)
    !       call def_budgetip_input(kgdsi)
    endif
#endif
```

1.1.5 get (Source File: baseforcing_module.F90)

Retrieves and interpolates model forcing

INTERFACE:

```
subroutine get()
```

USES:

```
use lisdrv_module, only: getforcing
```

CONTENTS:

```
call getf(getforcing())
```

1.1.6 time_interp (Source File: baseforcing_module.F90)

Computes temporal interpolation

INTERFACE:

```
subroutine time_interp()
```

USES:

```
use lisdrv_module,only :getforcing
```

CONTENTS:

```
call timeinterp(getforcing())
```

CONTENTS:

```
nmif = getnmif()
if(masterproc) then
    allocate(glbdata1(nmif,lis%d%glbngrid))
    allocate(glbdata2(nmif,lis%d%glbngrid))
else
    allocate(glbdata1(nmif,gdi(iam)))
    allocate(glbdata2(nmif,gdi(iam)))
endif
```

1.1.7 ld_getbaseforcing (Source File: baseforcing_module.F90)

Retrieves model forcing and invokes spatial and interpolation routines

INTERFACE:

```
subroutine ld_getbaseforcing()
```

USES:

```
use lisdrv_module, only: lis
```

CONTENTS:

```
#if ( defined OPENDAP )
    call get()
#else
    if ( masterproc ) then
        call get()
    endif
    call MPI_BCAST(lis%f%findtime1,1,MPI_INTEGER,0, &
                  MPI_COMM_WORLD, ier)
    call MPI_BCAST(lis%f%findtime2,1,MPI_INTEGER,0, &
                  MPI_COMM_WORLD, ier)
    if(lis%f%findtime1 ==1 .or. &
       lis%f%findtime2==1) then
        if(npes > 1) then
            call scatter_data()
        endif
    endif
#endif
    call time_interp()
```

1.1.8 scatter_data (Source File: baseforcing_module.F90)

Distributes the forcing data on compute nodes

INTERFACE:

```
subroutine scatter_data()
```

USES:

```
use lisdrv_module, only : lis
```

1.1.9 define_gds.F90 (Source File: define_gds.F90)

Assigns a grid definition section (GDS) array appropriate to the global resolution used.

REVISION HISTORY:

```
20 Jul 2001: Urszula Jambor; Initial code
12 Feb 2002: Urszula Jambor; Added latmax variable assignment
06 Mar 2002: Urszula Jambor; Added 1 & 1/2 degree resolution GDS arrays
```

INTERFACE:

```
subroutine define_gds ( lis )
```

USES:

```
use lis_module      ! LDAS non-model-specific 1-D variables
#if ( defined OPENDAP )
  use opendap_module, only : parm_nc, parm_nr, output_slat, output_nlat
#endif
  implicit none
```

ARGUMENTS:

```
type (lisdec):: lis
```

CONTENTS:

```
do i=1,200
  lis%d%kgds(i) = 0
end do
!-----
!      kgds(1) = 4 !Input grid type (4=Gaussian)
!      kgds(2) = 128 !Number of points on a lat circle
!      kgds(3) = 64 !Number of points on a meridian
!      kgds(4) = -87864 !Latitude of origin x1000
!      kgds(5) = 0 !Longitude of origin x1000
!      kgds(6) = 128 !8 bits (1 byte) related to resolution
! !(recall that 10000000 = 128), Table 7
!      kgds(7) = 87864 !Latitude of extreme point x1000
!      kgds(8) = -2812 !Longitude of extreme point x1000
!      kgds(9) = 2812 !N/S direction increment x1000
!      kgds(10) = 32 !(Gaussian) # lat circles pole-equator
!      kgds(11) = 64 !8 bit scanning mode flag (Table 8)
!-----
#if ( defined OPENDAP )
  nc = parm_nc
  nr = parm_nr
#else
  nc = lis%d%lnc
  nr = lis%d%lnr
```

```

#endif

select case (nc)
case ( 7200) ! 5km resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
  lis%d%kgds(4) = -59975
  lis%d%kgds(5) = -179975
  lis%d%kgds(6) = 128
  lis%d%kgds(7) = 89975
  lis%d%kgds(8) = 179975
  lis%d%kgds(9) = 50
  lis%d%kgds(10) = 50
  lis%d%kgds(11) = 64
  lis%d%kgds(20) = 255
  lis%f%latmax = 1800
case ( 2880 ) ! 1/8 degree resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
  lis%d%kgds(4) = -59939
  lis%d%kgds(5) = -179938
  lis%d%kgds(6) = 128
  lis%d%kgds(7) = 89938
  lis%d%kgds(8) = 179938
  lis%d%kgds(9) = 125
  lis%d%kgds(10) = 125
  lis%d%kgds(11) = 64
  lis%d%kgds(20) = 255
  lis%f%latmax = 720
case ( 1440 ) ! 1/4 degree resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
#if ( defined OPENDAP )
  lis%d%kgds(4) = output_slat
#else
  lis%d%kgds(4) = -59875
#endif
  lis%d%kgds(5) = -179875
  lis%d%kgds(6) = 128
#if ( defined OPENDAP )
  lis%d%kgds(7) = output_nlat
#else
  lis%d%kgds(7) = 89875
#endif
  lis%d%kgds(8) = 179875

```

```

lis%d%kgds(9) = 250
lis%d%kgds(10) = 250
lis%d%kgds(11) = 64
lis%d%kgds(20) = 255
lis%f%latmax = 360
case ( 720 ) ! 1/2 degree resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
  lis%d%kgds(4) = -59750
  lis%d%kgds(5) = -179750
  lis%d%kgds(6) = 128
  lis%d%kgds(7) = 89750
  lis%d%kgds(8) = 179750
  lis%d%kgds(9) = 500
  lis%d%kgds(10) = 500
  lis%d%kgds(11) = 64
  lis%d%kgds(20) = 255
  lis%f%latmax = 180
case ( 360 ) ! 1 degree resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
#if ( defined OPENDAP )
  lis%d%kgds(4) = output_slat
#else
  lis%d%kgds(4) = -59500
#endif
  lis%d%kgds(5) = -179500
  lis%d%kgds(6) = 128
#if ( defined OPENDAP )
  lis%d%kgds(7) = output_nlat
#else
  lis%d%kgds(7) = 89500
#endif
  lis%d%kgds(8) = 179500
  lis%d%kgds(9) = 1000
  lis%d%kgds(10) = 1000
  lis%d%kgds(11) = 64
  lis%d%kgds(20) = 255
  lis%f%latmax = 90
case ( 144 ) ! 2 x 2.5 degree resolution
  lis%d%kgds(1) = 0
  lis%d%kgds(2) = nc
  lis%d%kgds(3) = nr
  lis%d%kgds(4) = -60000
  lis%d%kgds(5) = -180000
  lis%d%kgds(6) = 128

```

```

lis%d%kgds(7) = 90000
lis%d%kgds(8) = 177500
lis%d%kgds(9) = 2500
lis%d%kgds(10) = 2000
lis%d%kgds(11) = 64
lis%d%kgds(20) = 255
lis%f%latmax = 46 !Actually, 45 + 45 + 1
case default
print *, "No valid global grid defined for given resolution"
print *, "columns: ", lis%d%lnc, " rows: ", lis%d%lnr
print *, "Stopping..."
call endrun
end select

```

1.2 Fortran: Module Interface *def_ipMod.F90* (Source File: *def_ipMod.F90*)

This module contains routines that precomputes weights and other parameters required for spatial interpolation of model forcing

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

INTERFACE:

```
module def_ipMod
implicit none
```

ARGUMENTS:

```

real, allocatable :: rlat0(:)
real, allocatable :: rlon0(:)
integer, allocatable :: n110(:)
integer, allocatable :: n120(:)
integer, allocatable :: n210(:)
integer, allocatable :: n220(:)
real, allocatable :: w110(:),w120(:)
real, allocatable :: w210(:),w220(:)
integer :: mi, mo, nn
integer :: nb3,nb4

```

1.2.1 *allocate_ip* (Source File: *def_ipMod.F90*)

Allocates memory for interpolation of model forcing data (GEOS and GDAS)

INTERFACE:

```
subroutine allocate_ip(N)
```

ARGUMENTS:

```
integer :: N
```

CONTENTS:

```
allocate(rlat0(n))
allocate(rlon0(n))
allocate(n110(n))
allocate(n120(n))
allocate(n210(n))
allocate(n220(n))
allocate(w110(n))
allocate(w120(n))
allocate(w210(n))
allocate(w220(n))
mo = n
nn = n
```

```
w110 = 0.0
w120 = 0.0
w210 = 0.0
w220 = 0.0
```

1.2.2 def_ip_input (Source File: *def_ipMod.F90*)

Calculates spatial variables required for interpolation of GEOS/GDAS model forcing

INTERFACE:

```
subroutine def_ip_input (kgdsi)
```

USES:

```
use spmdMod
use lisdrv_module, only:lis
#if ( defined OPEN DAP )
    use opendap_module
#endif
!INPUT ARGUMENTS:
integer :: kgdsi(200)
```

CONTENTS:

```
#if ( ! defined OPEN DAP )
    integer :: nroffset = 0
#endif
```

```

kgdso = lis%d%kgds
#if ( defined OPENDAP )
    mo = parm_nc*parm_nr
#else
    mo = lis%d%lnc*lis%d%lnr
#endif
!-----
! Calls the routines to decode the grid description and
! calculates the weights and neighbor information to perform
! spatial interpolation. This routine eliminates the need to
! compute these weights repeatedly during interpolation.
!-----
if(kgdso(1).ge.0) then
    call gdswiz(kgdso, 0,mo,fill,xpts,ypts,rlon0,rlat0,nn,0)
endif
call gdswiz(kgdsi,-1,nn,fill,xpts,ypts,rlon0,rlat0,nv,0)
do n=1,nn
    xi=xpts(n)
    yi=ypts(n)
    if(xi.ne.fill.and.yi.ne.fill) then
        i1=xi
        i2=i1+1
        j1=yi
        j2=j1+1
        xf=xi-i1
        yf=yi-j1
        n110(n)=ijkgs(i1,j1,kgdsi)
        n210(n)=ijkgs(i2,j1,kgdsi)
        n120(n)=ijkgs(i1,j2,kgdsi)
        n220(n)=ijkgs(i2,j2,kgdsi)
        if(min(n110(n),n210(n),n120(n),n220(n)).gt.0) then
            w110(n)=(1-xf)*(1-yf)
            w210(n)=xf*(1-yf)
            w120(n)=(1-xf)*yf
            w220(n)=xf*yf
        else
            n110(n)=0
            n210(n)=0
            n120(n)=0
            n220(n)=0
        endif
    else
        n110(n)=0
        n210(n)=0
        n120(n)=0
        n220(n)=0
    endif
enddo

```

```
#if ( defined OPENDAP )
!    mi = geos_nc*geos_nr
#else
!    mi = lis%f%ncold*lis%f%nrold
#endif
```

1.3 Fortran: Module Interface driverpardef_module.F90 (Source File: driverpardef_module.F90)

This module contains routines that defines MPI derived data types for LIS driver specific variables

REVISION HISTORY:

06 Oct 2003; Sujay Kumar Initial Specification

INTERFACE:

```
module driverpardef_module
```

USES:

```
use tile_module
use lis_module
use grid_module
use spmdMod
implicit none
```

ARGUMENTS:

```
integer:: MPI_TILE_STRUCT !MPI derived type for tile$-$module
integer:: MPI_GRID_STRUCT !MPI derived type for grid$-$module
integer:: MPI_LD_STRUCT !MPI derived type for lisdomain
integer:: MPI_LF_STRUCT !MPI derived type for lisforcing
integer:: MPI_LP_STRUCT !MPI derived type for lisparameters
integer:: MPI_LT_STRUCT !MPI derived type for listime
integer:: MPI_LO_STRUCT !MPI derived type for lisoutput
integer:: MPI_LA_STRUCT !MPI derived type for lisassimil
```

1.3.1 def_driverpar_structs (Source File: driverpardef_module.F90)

Routine that defines MPI derived data types

INTERFACE:

```
subroutine def_driverpar_structs()
```

1.3.2 elevadjust.F90 (Source File: elevadjust.F90)

Corrects Temperature, Pressure, Humidity and Longwave Radiation values for differences in elevation between EDAS and LDAS grids.

REVISION HISTORY:

```

11 Apr 2000: Brian Cosgrove; Initial Code
12 May 2000: Brian Cosgrove; Corrected for zero humidities
09 Aug 2000: Brian Cosgrove; Corrected program so that
              it only performs calculations if both
              the elevation difference file and the forcing
              data file (use temperature data as check for all
              fields) contain defined values
25 Jan 2001: Matt Rodell; Compute number of input and output
grid points, use to allocate local arrays
27 Feb 2001: Brian Cosgrove; Added statement to check for use of
              catchment data so that correct elevation correction
              files is used
15 Mar 2001: Jon Gottschalck; if-then to handle negative vapor
pressures in long wave correction
15 Mar 2001: Matt Rodell; merge NLDAS and GLDAS versions
14 Nov 2003: Sujay Kumar; Adopted in LIS

```

INTERFACE:

```
subroutine elevadjust(t,f,fforce,force_tmp,force_hum,force_lwd, &
                     force_prs)
```

USES:

```
use lisdrv_module, only: tile
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: f, t
```

OUTPUT PARAMETERS:

```
real, intent(inout) :: fforce,force_tmp,force_hum,&
                     force_lwd,force_prs
```

CONTENTS:

```

grav = 9.81
rdry = 287.
lapse = -0.0065
tcforce=force_tmp+(lapse*tile(t)%elev)
tbar=(force_tmp+tcforce)/2.
pcforce=force_prs/(exp((grav*tile(t)%elev)/(rdry*tbar)))
if (force_hum .eq. 0) force_hum=1e-08
ee=(force_hum*force_prs)/0.622

```

```

esat=611.2*(exp((17.67*(force_tmp-273.15))/&
    ((force_tmp-273.15)+243.5)))
qsat=(0.622*esat)/(force_prs-(0.378*esat))
rh=(force_hum/qsat)*100.
fesat=611.2*(exp((17.67*(tcforce-273.15))/
    ((tcforce-273.15)+243.5)))
fqsat=(0.622*fesat)/(pcforce-(0.378*fesat))
hcforce=(rh*fqsat)/100.
fe=(hcforce*pcforce)/0.622
mee=ee/100.
mfe=fe/100.

!-----
! correct for negative vapor pressure at very low temperatures at
! high latitudes
!-----

if (mee .le. 0) mee = 1e-08
if (mfe .le. 0) mfe = 1e-08
emiss =1.08*(1-exp(-mee** (force_tmp/bb)))
femiss =1.08*(1-exp(-mfe** (tcforce/bb)))
ratio=(femiss*(tcforce**4))/(emiss*(force_tmp**4))
lcforce=force_lwd*ratio

select case (f)
case(1)
    fforce=tcforce
case(2)
    fforce=hcforce
case(4)
    fforce=lcforce
case(7)
    fforce=pcforce
case default
    print*, "not a valid forcing type for elevation adjustment"
    call endrun
end select
return

```

1.3.3 endrun.F90 (Source File: endrun.F90)

Routine to be called to terminate the program. This routines flushes the output streams and aborts the mpi processes.

REVISION HISTORY:

```

14Nov02 Sujay Kumar Initial Specification
#include <misc.h>

```

INTERFACE:

```
subroutine endrun
```

USES:

```
use mpishorthand, only: MPI_COMM_WORLD
```

CONTENTS:

```
write(6,*)'endrun is being called'
call flush( 6 )    ! Flush all output to standard output
#ifndef (defined SPMD)
call mpi_abort (MPI_COMM_WORLD, 1)
#else
call abort
#endif
```

1.4 Fortran: Module Interface grid_module.F90 (Source File: grid_module.F90)

LIS non-model-specific grid variables only.

FORCING() ARRAY:

1. T 2m Temperature interpolated to 2 metres [K]
2. q 2m Instantaneous specific humidity interpolated to 2 metres[kg/kg]
3. radswg Downward shortwave flux at the ground [W/m²]
4. lwgdwn Downward longwave radiation at the ground [W/m²]
5. u 10m Instantaneous zonal wind interpolated to 10 metres [m/s]
6. v 10m Instantaneous meridional wind interpolated to 10 metres[m/s]
7. ps Instantaneous Surface Pressure [Pa]
8. preacc Total precipitation [mm/s]
9. precon Convective precipitation [mm/s]
10. albedo Surface albedo (0-1)

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial code
11 Apr 2000: Brian Cosgrove; Added Forcing Mask variables
23 Feb 2001: Urszula Jambor; Added GEOS & GDAS forcing variables
27 Feb 2001: Brian Cosgrove; Added Catchment forcing data variables
23 Mar 2001: Jon Radakovich; Added variables for PSAS assimilation
04 Sep 2001: Brian Cosgrove; Added variabes for humidity, precip,par
            brightness temp,precip mask, removed awips2lis and
            pinker2lis variables, GRIB interp. package used now
15 Oct 2001: Jesse Meng; Revised doc block with forcing array definition
15 Oct 2001: Jesse Meng; Added oblwdatal and oblwdatal2
14 Nov 2002: Sujay Kumar; Optimized version of grid_module
```

INTERFACE:

```
module grid_module
  implicit none
  public griddec
```

ARGUMENTS:

```
type griddec
  real    :: lat          !latitude of grid point
  real    :: lon          !longitude of grid point
  real    :: forcing(10)   !interpolated LIS forcing array
  real    :: fgrd(13)     !fraction of vegetation class in grid
end type griddec
```

1.5 Fortran: Module Interface grid_spmdMod.F90 (Source File: grid_spmdMod.F90)

This module computes domain decomposition on the grid domain

REVISION HISTORY:

14 Nov 2002 Sujay Kumar Initial Specification

INTERFACE:

```
module grid_spmdMod
```

USES:

```
use spmdMod
```

ARGUMENTS:

```
integer, allocatable :: gdi(:), gdisp(:)
integer, allocatable :: g2di(:), g2disp(:)
```

1.5.1 allocate_gdd (Source File: grid_spmdMod.F90)

Allocates memory for arrays that contain domain decomposition information

INTERFACE:

```
subroutine allocate_gdd()
```

DESCRIPTION:

Allocates memory for arrays that contain domain decomposition information

```
allocate(gdi(0:npes-1))
allocate(gdisp(0:npes-1))
allocate(g2di(0:npes-1))
allocate(g2disp(0:npes-1))
```

1.5.2 grid_spmd_init (Source File: grid_spmdMod.F90)

Computes domain decomposition based on the number of processors

INTERFACE:

```
subroutine grid_spmd_init(tile,nch,nmif,ngrid)
```

USES:

```
use tile_module
use tile_spmdMod, only : displs
!INPUT ARGUMENTS:
integer :: nch, ngrid, nmif
!OUTPUT ARGUMENTS:
type(tileddec)::tile(nch)
```

CONTENTS:

```
gdisp(0) = 0
do p=1, npes-1
    gdisp(p) = tile(displs(p))%index
enddo
do p = 0 , npes-2
    gdi(p) = gdisp(p+1)-gdisp(p)
enddo
gdi(npes-1) = ngrid - gdisp(npes-1)
do p=0, npes-1
    g2di(p) = gdi(p)*nmif
enddo
g2disp(0) = 0
do p=1, npes-1
    g2disp(p) = g2disp(p-1)+g2di(p-1)
enddo
```

1.5.3 lisdrv.F90 - Main program for LIS (Source File: lisdrv.F90)

Main driver program for LIS. It initializes the boundary conditions, allocates memory for the required variables, sets up appropriated model parameters, performs the I/O for forcing and land models, in addition to calling the appropriate land model over different time steps and geographical domains.

1.6 Function calls for main routines:

The function calls are:

LIS_domain_init Initializes the domain variables

LIS_allocate_memory Allocates memory for modules, variables

LIS_lsm_init Initializes land surface model run parameters

LIS_baseforcing_init Initializes model forcing variables

LIS_readrestart Reads the restart files

LIS_setuplsm Completes initialization of the land surface model

LIS_ticktime Manages the advancement of time

LIS_endofrun Checks if the end of simulation is reached

LIS_get_base_forcing Reads, interpolates model forcing

LIS_force2tile Transfers grid forcing to model tiles.

LIS_lsm_main Executes land surface model runs.

LIS_lsm_output Writes land surface model output

LIS_write_restart Writes restart files

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

USES:

```
use precision
use lisdrv_module
use lsm_module
use baseforcing_module
use obsprecipforcing_module
use obsradforcing_module
use spmdMod
```

CONTENTS:

```
call LIS_domain_init
call LIS_allocate_memory
call LIS_lsm_init
call LIS_allocate_lsm
call LIS_baseforcing_init
call LIS_obsprecipforcing_init
call LIS_obsradforcing_init
call LIS_readrestart
call LIS_setuplsm
do while (.NOT. LIS_endofrun())
    call LIS_ticktime
    call LIS_setDynlsm
```

```

call LIS_get_base_forcing
call LIS_get_obsprecip_forcing
call LIS_get_obsrad_forcing
call LIS_force2tile
call LIS_lsm_main
call LIS_lsm_output
call LIS_writerestart
enddo

```

1.7 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90)

Main program for LIS This module contains interfaces and subroutines that control program execution.

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

USES:

```

use lis_module
use grid_module
use time_manager
use tile_spmdMod
use tile_module

```

INTERFACE:

```

interface LIS_domain_init
    module procedure ld_domain_init
end interface

```

1.7.1 ld_domain_init (Source File: lisdrv_module.F90)

Calls routines to read the card file, and initialize the time manager

INTERFACE:

```

subroutine ld_domain_init()

```

CONTENTS:

```

#ifndef SPMD
    call spmd_init()
#endif
    if ( masterproc ) then
        call readcard(lis)
!-----
! we have read in the time parameters. Use them to initialize ESMF
! time manager
!-----
        call setup_timeMgr()
        lis%t%endtime = 0
    endif

```

1.7.2 setup_timeMgr (Source File: lisdrv_module.F90)

Initializes the ESMF time manager

INTERFACE:

```
subroutine setup_timeMgr()
```

CONTENTS:

```

dtime = lis%t%ts
start_ymd=lis%t%syr*10000 + lis%t%smo*100 + lis%t%sda
start_tod=(lis%t%shr*3600)+lis%t%sss
stop_ymd =lis%t%eyr*10000 + lis%t%emo*100 + lis%t%eda
stop_tod =(lis%t%ehr*3600)+lis%t%ess
call timemgr_init()
print*, 'time manager initialized..'

```

1.7.3 LIS_ticktime (Source File: lisdrv_module.F90)

Uses the ESMF time manager to handle model timestepping.

INTERFACE:

```
subroutine LIS_ticktime()
```

USES:

```
use driverpardef_module, only: MPI_LT_STRUCT
```

CONTENTS:

```

if(masterproc) then
    call advance_timestep()
    lis%t%tscount = get_nstep()
    call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
    call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
    call updatetime(lis%t) !Updates LDAS variables.
endif
#if (defined SPMD)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
#endif

```

1.7.4 LIS_allocate_memory (Source File: lisdrv_module.F90)

Allocates memory for the domain variables, initializes MPI data structures, and computes domain decomposition

INTERFACE:

```
subroutine LIS_allocate_memory()
```

CONTENTS:

```

if ( masterproc ) then
    nc = getnc()
    nr = getnr()
    maxt = getmaxt()
    call setnch(nc, nr, maxt)
endif
#if ( defined OPENDAP )
    call MPI_BCAST(lis%d%lnc, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%lnr, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%nt, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%mina, 1, MPI_REAL, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%maxt, 1, MPI_INTEGER, 0, &
                  MPI_COMM_WORLD, ierr)
!    call MPI_BCAST(lis%p%koster, 1, MPI_INTEGER, 0, &
!                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%mfile, 50, MPI_CHARACTER, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%vfile, 50, MPI_CHARACTER, 0, &
                  MPI_COMM_WORLD, ierr)

```

```

#endif
#if ( defined OPENDAP )
    call maketiles_gds()
#else
    call maketiles_nongds()
#endif
#if (defined SPMD)
    call def_driverpar_structs()
    call MPI_BCAST(lis%d, 1, MPI_LD_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%f, 1, MPI_LF_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p, 1, MPI_LP_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%o, 1, MPI_LO_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%a, 1, MPI_LA_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
                  MPI_COMM_WORLD, ierr)
#endif ( ! defined OPENDAP )
    lis%d%ngrid = lis%d%glbngrid
    lis%d%nch   = lis%d%glbnch
#endif

call allocate_tileddd
if(masterproc) then
    call tile_spmd_init(tile, lis%d%glbnch, lis%f%nmif)
endif
call spread_tdds()
if(.NOT.masterproc) then
    allocate(tile(di_array(iam)))
endif
if(npes>1) then
    call MPI_SCATTERV(tile,di_array,displs, &
                      MPI_TILE_STRUCT,tile,di_array(iam),MPI_TILE_STRUCT, &
                      0,MPI_COMM_WORLD,ierr)
endif
call allocate_gdd
if(masterproc) then
    call grid_spmd_init(tile,lis%d%glbnch, &
                        lis%f%nmif, lis%d%glbngrid)
endif
call spread_gdds()
if(.NOT.masterproc) then
    allocate(grid(gdi(iam)))
endif
if(npes >1) then

```

```

call MPI_SCATTERV(grid,gdi,gdisp, &
                  MPI_GRID_STRUCT,grid,gdi(iam),MPI_GRID_STRUCT, &
                  0,MPI_COMM_WORLD,ierr)
endif
call dist_gindex()
if(masterproc) then
  call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
  call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
  call updatetime(lis%t)
endif
#if (defined SPMD)
  call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
                 MPI_COMM_WORLD, ierr)
#endif
#endif

```

1.7.5 setnch (Source File: lisdrv_module.F90)

Computes an estimate of the total number of tiles

INTERFACE:

```
subroutine setnch(nc, nr, maxt)
```

ARGUMENTS:

```

integer::nc
integer::nr
integer::maxt
```

CONTENTS:

```
lis%d%glnch = nc*nr*maxt !!may be too big
```

1.7.6 getdomain (Source File: lisdrv_module.F90)

Returns the domain resolution

INTERFACE:

```
function getdomain() result(d)
```

ARGUMENTS:

```
integer :: d
```

CONTENTS:

```
d = lis%d%domain
```

1.7.7 getlsm (Source File: lisdrv_module.F90)

Returns which land surface model is executed

INTERFACE:

```
function getlsm() result(lsmno)
```

ARGUMENTS:

```
integer :: lsmno
```

CONTENTS:

```
lsmno = lis%d%lsm
```

1.7.8 getnch (Source File: lisdrv_module.F90)

Returns the number of model tiles

INTERFACE:

```
function getnch() result(n)
```

ARGUMENTS:

```
integer :: n
```

CONTENTS:

```
n = lis%d%glbnch
```

1.7.9 getnc (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnc() result(ncol)
```

ARGUMENTS:

```
integer :: ncol
```

CONTENTS:

```
ncol = lis%d%lnc
```

1.7.10 getnr (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnr() result(nrow)
```

ARGUMENTS:

```
integer :: nrow
```

CONTENTS:

```
nrow = lis%d%lnr
```

1.7.11 getmaxt (Source File: lisdrv_module.F90)

Returns the maximum number of tiles

INTERFACE:

```
function getmaxt() result(maxt)
```

ARGUMENTS:

```
integer :: maxt
```

CONTENTS:

```
maxt = lis%d%maxt
```

1.7.12 getforcing (Source File: lisdrv_module.F90)

Returns the type of forcing used

INTERFACE:

```
function getforcing() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%force
```

1.7.13 getnmif (Source File: lisdrv_module.F90)

Returns the number of forcing variables for model initialization

INTERFACE:

```
function getnmif() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%nmif
```

1.7.14 LIS_endofrun (Source File: lisdrv_module.F90)

Returns if the end of simulation has reached

INTERFACE:

```
function LIS_endofrun() result(finish)
```

ARGUMENTS:

```
logical :: finish
integer :: ierr
```

CONTENTS:

```
if(masterproc) then
    finish = is_last_step()
endif
#if (defined SPMD)
    call MPI_BCAST(finish, 1, MPI_LOGICAL, 0, &
                  MPI_COMM_WORLD, ierr)
#endif
```

1.7.15 dist_gindex (Source File: lisdrv_module.F90)

Distributes the mask indices on compute nodes for a GDS-based execution

INTERFACE:

```
subroutine dist_gindex
```

USES:

```
use grid_spmdMod
```

```
implicit none
```

ARGUMENTS:

```
integer :: findex, lindex, nr_count, t
integer :: ierr, status(MPI_STATUS_SIZE)
integer :: grid_offset, grid_lb, grid_ub
integer :: i, j
```

CONTENTS:

```
#if ( defined OPENDAP )
  if ( npes > 1 ) then
    if ( masterproc ) then
      do t = 1, npes-1
        findex = tile( gdisp(t) + 1 )%row
        lindex = tile( gdisp(t) + gdi(t) )%row
        nr_count = (lindex-findex+1)
        print*, 'DBG: lisdrv_module -- ', &
          't, findex,lindex,lis%d%gnc,nr_count', &
          t, findex,lindex,lis%d%gnc,nr_count, ' (', iam, ')'
        call mpi_send(nr_count,1,MPI_INTEGER,t,t, &
          MPI_COMM_WORLD,ierr)
        call mpi_send(glbgindex(:,findex:lindex), &
          lis%d%gnc*nr_count, &
          MPI_INTEGER,t,t,MPI_COMM_WORLD,ierr)
      enddo
    else
      call mpi_recv(nr_count,1,MPI_INTEGER,0,MPI_ANY_TAG, &
        MPI_COMM_WORLD,status,ierr)
      allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
      call check_error(ierr,'lisdrv_module', &
        'Error allocating gindex.',iam)
      call mpi_recv(gindex,lis%d%gnc*nr_count,MPI_INTEGER,0, &
        MPI_ANY_TAG, &
        MPI_COMM_WORLD,status,ierr)
    endif
  endif

  if ( masterproc ) then
    findex = tile( gdisp(0) + 1 )%row
    lindex = tile( gdisp(0) + gdi(0) )%row
    nr_count = (lindex-findex+1)
    print*, 'DBG: lisdrv_module -- ', &
      'findex,lindex,lis%d%gnc,nr_count', &
      findex,lindex,lis%d%gnc,nr_count, ' (', iam, ')'
    allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
```

```

call check_error(ierr,'lisdrv_module', &
                 'Error allocating gindex.',iam)
print*, 'DBG: lisdrv_module -- size(gindex),size(glbginde)', &
        size(gindex(1:lis%d%gnc,1:nr_count)), &
        size(glbginde(1:lis%d%gnc,findex:lindex))
gindex = glbginde(:,findex:lindex)
endif

grid_lb = gdisp(iam) + 1
grid_ub = gdisp(iam) + gdi(iam)
grid_offset = gdisp(iam)
do j = 1, nr_count
  do i = 1, lis%d%gnc
    if ( gindex(i,j) /= -1 ) then
      if ( ( gindex(i,j) < grid_lb ) .or. &
           ( gindex(i,j) > grid_ub ) ) then
        gindex(i,j) = - 1
      else
        gindex(i,j) = gindex(i,j) - grid_offset
      endif
    endif
  enddo
enddo
#else
  if ( masterproc ) then
    allocate(gindex(lis%d%lnc,lis%d%lnr),stat=ierr)
    call check_error(ierr,'lisdrv_module', &
                     'Error allocating gindex.',iam)
    gindex=glbginde
  endif
#endif

```

1.8 Fortran: Module Interface lis_module.F90 (Source File: lis_module.F90)

Module for LDAS variable specification. This file will contain no tile space or grid space variables.

REVISION HISTORY:

- 15 Oct 1999: Paul Houser; Initial code
- 4 Apr 2000: Jeffrey Walker; Added some catchment model variables
- 11 Apr 2000: Brian Cosgrove; Elevation correction and Forcing Mask variables added
- 6 Jun 2000: Jon Radakovich; Updated for new version of CLM
- 23 Feb 2001: Urszula Jambor; Updated for GEOS & GDAS forcing in GLDAS
- 27 Feb 2001: Brian Cosgrove; Added Catchment forcing data variables

15 Mar 2001: Jon Gottschalck; Updated for GDAS initialization of Mosaic
 12 Apr 2001: Urszula Jambor; Added domain, lsm, & force namefile paramters
 30 Apr 2001: Jon Radakovich; Update for PSAS temperature assimilation
 17 Jul 2001: Jon Gottschalck; Update for global precipitation variables
 30 Jul 2001: Matt Rodell; Add new soil parameter variables
 05 Sep 2001: Brian Cosgrove; Add variables for PAR and BRITMP, remove
 1/4 to 1/8 interp variables
 15 Oct 2001: Jesse Meng; Replace agrmet flag by agrmetsw and agrmetlw
 27 Nov 2001: Jon Gottschalck; Added variables for AVHRR LAI data
 07 Dec 2001: Urszula Jambor; Added LDAS_KGDS array
 03 Feb 2002: Jon Gottschalck; Added Koster tilespace variables
 05 Feb 2002: Brian Cosgrove; Added NLDAS 11 Layer soil class file from Yun Duan
 and ftype variable to indicate NLDAS ETA forcing source used
 08 Feb 2002: Urszula Jambor; Added latmax for AGRMET use.
 15 Apr 2002: Urszula Jambor; Added ECMWF forcing options.
 28 Apr 2002: Kristi Arsenault; Added NOAH LSM parameters and code
 14 Nov 2002; Sujay Kumar; Optimized version for LIS
 14 Oct 2003; Sujay Kumar; Removed LSM specific variables.

INTERFACE:

```
module lis_module
```

USES:

```
use time_module
implicit none
```

ARGUMENTS:

```

type lisdomain
  INTEGER :: NCH          !actual number of tiles
  INTEGER :: LSM          !Land surface model (2=CLM,4=NOAH)
  INTEGER :: SOIL         !Soil parameter scheme (1=original veg-based, 2=Reynolds soils)

  INTEGER :: GLBNCH       !actual global number of tiles
  INTEGER :: NGRID        !actual number of grids
  INTEGER :: GLBNGRID     !actual global number of grids
  INTEGER :: DOMAIN       !Model domain, (1=NLDAS, 2=GLDAS)
  integer :: GNC          !Global array (different if subsetting is used)
  integer :: GNR          !Global array (different if subsetting is used)
  INTEGER :: LNC          !Local Number of Columns in Grid
  INTEGER :: LNR          !Local Number of Rows in Grid
  INTEGER :: MAXT         !Maximum tiles per grid
  INTEGER :: KGDS(200)     !Grid definition array
  REAL :: MINA            !Min grid area for tile (%)
  REAL :: UDEF            !Undefined value
end type lisdomain

type lisforcing
```

```

INTEGER :: FORCE          !Forcing data type (1=GDAS,2=GEOS)
INTEGER :: ECOR            !Use elevation correction
INTEGER :: NFORCE
INTEGER :: NF               !Number of forcing variables
INTEGER :: NMIF             !Number of forcing variables for model initialization option
INTEGER :: RSTFLAG           !0=Use only 1 forcing time, 1=find two forcing times upon restat
INTEGER :: GRIDCHANGE
INTEGER :: latmax            !Per hemisphere, for AGRMET intepolation

INTEGER :: SHORTFLAG         !Shortwave radiation source flag
                           !0=No radiation
                           !1=Instantaneous SW
                           !2=Time Averaged SW
INTEGER :: LONGFLAG          !Longwave radiation source flag
                           !0=No radiation
                           !1=Instantaneous LW
                           !2=Time Averaged LW
integer :: findtime1,findtime2
integer :: findagrttime1,findagrttime2
integer :: f00_flag, f06_flag
integer :: gpcpsrc      !Global precipitation flags
integer :: radsrc
end type lisforcing

type lisparameters
    INTEGER :: LAI              !LAI data source (1=original,
                               !2=AVHRR satellite data
                               !3=MODIS satellite data)
    INTEGER :: NT                !Number of Vegetation Types
    INTEGER :: VCLASS            !Vegetation Classification (1=UMD)
    INTEGER :: KOSTER            ! Flag to use Koster tilespace at 2.0x2.5 degrees
    integer :: laiflag            !Satellite LAI Time
    CHARACTER*50 :: MFILE          !land/water mask file for MODELLING (AVHRR)
    CHARACTER*50 :: VFILE          !vegetation classification file (AVHRR) !
    CHARACTER*40 :: SAFILE          !Sand fraction map file
    CHARACTER*40 :: CLFILE          !Clay fraction map file
    CHARACTER*40 :: AVHRRDIR        !AVHRR Data directory
    CHARACTER*40 :: MODISDIR        !MODIS Data directory
    CHARACTER*40 :: ISCFILE          !Soil color map file
    character*40 :: elevfile
    REAL*8   :: LAITIME          !Satellite LAI Time

end type lisparameters

type lisoutput
    INTEGER :: WFOR              !Write Forcing (0=no,1=yes)
    INTEGER :: WTIL               !Write tile space data (0=no, 1=yes)
    INTEGER :: WOUT                !Output format option (1-binary, 2-grib)

```

```

INTEGER :: WSINGLE      !Write one variable per file
INTEGER :: WPARAM        !Write parameter file output
INTEGER :: EXPCODE       !3 Digit experiment code
INTEGER :: STARTCODE     !0=restart date, 1=card date
INTEGER :: FORopen
INTEGER :: NUMOUTF      !Counts number of output times for forcing data
INTEGER :: FIDGM,RCGM,FIDTM,RCTM
CHARACTER*40 :: ODIR      !Output Data Base Directory
CHARACTER*40 :: DFILE      !runtime diagnostics file
end type lisoutput

type lisassimil
    integer :: rpsas, rbias,ribc,rdbc,rsdbc
end type lisassimil
type lisdec
    type(lisdomain)   :: d
    type(lisforcing)   :: f
    type(lisparameters) :: p
    type(listime)      :: t
    type(lisoutput)     :: o
    type(lisassimil)   :: a
end type lisdec

```

1.9 Fortran: Module Interface lsm_module.F90 (Source File: lsm_module.F90)

This module contains interfaces and subroutines that control land surface model initialization, execution, reading and writing of restart files and other relevant land surface model computations.

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

INTERFACE:

```
module lsm_module
```

1.9.1 LIS_lsm_init (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_lsm_init
    module procedure init
end interface
```

1.9.2 LIS_allocate_lsm (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_allocate_lsm
```

1.9.3 LIS_setuplsm (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_setuplsm
```

1.9.4 LIS_lsm_main (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_lsm_main
```

1.9.5 LIS_force2tile (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_force2tile
```

1.9.6 LIS_readrestart (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_readrestart
```

1.9.7 LIS_writerestart (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_writerestart
```

1.9.8 LIS_lsm_output (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_lsm_output
```

1.9.9 LIS_setDynlsm (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_setDynlsm
```

INTERFACE:

```
subroutine init()
```

DESCRIPTION:

Setup functions for each land surface model

USES:

```
use lsm_pluginMod
```

CONTENTS:

```
call lsm_plugin
```

1.9.10 lsm_tile_allocate (Source File: lsm_module.F90)

Allocates memory for land surface model variables

INTERFACE:

```
subroutine lsm_tile_allocate()
```

USES:

```
use lisdrv_module, only: lis
#if ( defined OPENDAP )
    use opendap_module
#endif
```

CONTENTS:

```
#if ( defined OPENDAP )
    call opendap_init()
#endif
    if ( lis%o%wsingle == 1 ) then
        call lsmini(lis%d%lsm,lis%d%nch)
    else
        call lsmini(lis%d%lsm,lis%d%glnch)
    endif
```

1.9.11 lsm_setup (Source File: lsm_module.F90)

Completes land surface model initilaization

INTERFACE:

```
subroutine lsm_setup()
```

USES:

```
use lisdrv_module, only : lis
use tile_spmdMod, only : masterproc
```

CONTENTS:

```
call lsmsetup(lis%d%lsm)
```

1.9.12 run_lsm (Source File: lsm_module.F90)

Executes land surface model runs

INTERFACE:

```
subroutine run_lsm()
```

USES:

```
use lisdrv_module, only: lis
```

CONTENTS:

```
call lsrun(lis%d%lsm)
```

1.9.13 lsm_readrestart (Source File: lsm_module.F90)

Reads restart files

INTERFACE:

```
subroutine lsm_readrestart()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmrestart(lis%d%lsm,1)
```

1.9.14 write_output (Source File: lsm_module.F90)

Writes output of land surface model runs

INTERFACE:

```
subroutine write_output()
```

USES:

```
use lisdrv_module, only: lis
```

CONTENTS:

```
call lsmoutput(lis%d%lsm)
```

1.9.15 setDynParams (Source File: lsm_module.F90)

Updates time dependent land surface model parameters

INTERFACE:

```
subroutine setDynParams()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmdynsetup(lis%d%lsm)
```

1.9.16 lsm_f2t (Source File: lsm_module.F90)

Transfers grid forcing to model tiles.

INTERFACE:

```
subroutine lsm_f2t()
```

USES:

```
use lisdrv_module, only: lis, grid,tile
use tile_spmdMod
use grid_spmdMod, only : gdisp
```

CONTENTS:

```
if(lis%f%ecor .eq. 1) then
    do t=1,di_array(iam)
        index = tile(t)%index-gdisp(iam)
        do f=1,lis%f%nforce
            force_tmp = grid(index)%forcing(1)
            force_hum = grid(index)%forcing(2)
            force_lwd = grid(index)%forcing(4)
            force_prs = grid(index)%forcing(7)
            if (f .eq. 1 .or. f .eq. 2 .or. f .eq. 4 .or. f .eq. 7) then
                call elevadjust(t,f,fforce,force_tmp,force_hum,force_lwd, &
                               force_prs)
                grid(index)%forcing(f)=fforce
            endif
        enddo
    enddo
end if
do t=1, di_array(iam)
    index = tile(t)%index -gdisp(iam)
    call lsmf2t(lis%d%lsm, t, grid(index)%forcing)
enddo
```

1.9.17 lsm_writerestart (Source File: lsm_module.F90)

Writes restart files

INTERFACE:

```
subroutine lsm_writerestart()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmwrst(lis%d%lsm)
```

1.9.18 maketiles_gds() (Source File: maketiles_gds.F90)

This primary goal of this routine is to determine tile space for GDS based I/O

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Major F90 and major structure revision
3 Jan 2000: Minor T=0 bug fix, should have no effect on output
8 Mar 2000: Brian Cosgrove; Initialized FGRD to 0 For Dec Alpha Runs
22 Aug 2000: Brian Cosgrove; Altered code for US/Mexico/Canada Mask
04 Feb 2001: Jon Gottschalck; Added option to read and use Koster tile space

```

INTERFACE:

```

subroutine maketiles_gds()
#ifndef ( defined OPENDAP )

```

USES:

```

use lisdrv_module, only: lis, grid, glbgindex, tile
use grid_module
use spmdMod

```

CONTENTS:

```

#ifndef ( defined ABSOFT )
    character*15 :: home
    home = '/home/jim/'
#else
    character*2 :: home
    home = './'
#endif

lis%d%gnc = lis%d%lnc
lis%d%gnr = lis%d%lnr

allocate(revcnts2(0:npes-1))
allocate(displs2(0:npes-1))

lnr = lis%d%gnr / npes
lnr_last = lis%d%gnr - lnr*(npes-1)

displs2(0) = 0
do t = 1, npes-1
    displs2(t) = displs2(t-1) + lnr*lis%d%gnc
enddo

do t = 0, npes-2
    revcnts2(t) = lnr*lis%d%gnc
enddo

```

```

recvcnts2(npes-1) = lnr_last*lis%d%gnc

wlon = 1
elon = lis%d%gnc

slat = iam*lnr + 1
if ( iam == npes-1 ) then
    nlat = lis%d%gnr
else
    nlat = (iam+1)*lnr
endif

if ( iam == npes-1 ) then
    lnr = lnr_last
endif

write(cslat, '(i4)') slat
write(cnlat, '(i4)') nlat
write(cwlon, '(i4)') wlon
write(celon, '(i4)') elon
write(ciam, '(i3)') iam

print*,'DBG: maketiles_gds -- cslat ', cslat, ', (', iam, ',)'
print*,'DBG: maketiles_gds -- cnlat ', cnlat, ', (', iam, ',)'
print*,'DBG: maketiles_gds -- cwlon ', cwlon, ', (', iam, ',)'
print*,'DBG: maketiles_gds -- celon ', celon, ', (', iam, ',)'
print*,'DBG: maketiles_gds -- ciam ', ciam, ', (', iam, ',)'

lis%p%myfile = trim(home)//trim(adjustl(ciam))//''//lis%p%myfile
lis%p%vfile = trim(home)//trim(adjustl(ciam))//''//lis%p%vfile

allocate(lat(lis%d%gnc,lnr), stat=ierr)
call check_error(ierr,'Error allocating lat.',iam)

allocate(lon(lis%d%gnc,lnr), stat=ierr)
call check_error(ierr,'Error allocating lon.',iam)

allocate(fgrd(lis%d%gnc,lnr,lis%p%nt), stat=ierr)
call check_error(ierr,'Error allocating fgrd.',iam)

allocate(mask(lis%d%gnc, lnr), stat=ierr)
call check_error(ierr,'Error allocating mask.',iam)

allocate(pveg(lis%d%gnc,lnr,lis%p%nt), stat=ierr)
call check_error(ierr,'Error allocating pveg.',iam)

allocate(tsum(lis%d%gnc, lnr), stat=ierr)
call check_error(ierr,'Error allocating tsum.',iam)

```

```

allocate(veg(lis%d%gnc, lnr,lis%p%nt), stat=ierr)
call check_error(ierr,'Error allocating veg.',iam)

tsum = 0.0

nchp = lis%d%glbnch
print*, 'MSG: maketiles -- Retrieving UMD mask file ', &
trim(lis%p%mfile), ' (',iam,')'
call system("opendap_scripts/getmask.pl "//ciam//" //&
trim(lis%p%mfile)//" "// &
cslat//" //cnlat//" //cylon//" //celon)
print*, 'MSG: maketiles -- Reading ',trim(lis%p%mfile), &
' (',iam,')'
open(30,file=lis%p%mfile,form='unformatted',status='old')
read(30) lat
read(30) lon
read(30) mask
close(30)
print*, 'MSG: maketiles -- Done reading ',trim(lis%p%mfile), &
' (',iam,')'
!-----
! Select which tile-space veg. info to use (UMD or Koster)
!-----
! if (lis%p%koster .lt. 1) then           ! Use original UMD indexing
  print*, 'MSG: maketiles -- Retrieving UMD veg file ', &
trim(lis%p%vfile), ' (',iam,')'
  call system("opendap_scripts/getveg.pl "//ciam//" //&
trim(lis%p%vfile)//" "// &
cslat//" //cnlat//" //cylon//" //celon)
  print*, 'MSG: maketiles -- Reading ',trim(lis%p%vfile), &
' (',iam,')'
  open(98,file=lis%p%vfile,form='unformatted')
  read(98) lat
  read(98) lon
  do t = 1, lis%p%nt
    read(98) veg(:,:,t)
  enddo
  do r=1,lnr
    do c=1,lis%d%gnc
      isum=0.0
      do t=1,lis%p%nt
        isum=isum+veg(c,r,t)
      enddo
      do t=1,lis%p%nt
        fgrd(c,r,t)=0.0
        if(isum.gt.0) fgrd(c,r,t)=veg(c,r,t)/isum
      enddo
    enddo
  enddo
enddo

```

```

        enddo
    enddo
    close(98)
    print*, 'MSG: maketiles -- Done reading ', trim(lis%p%vfile), &
        ' (',iam,')
! endif
!-----
! Exclude tiles with MINA (minimum tile grid area),
! normalize remaining tiles to 100%
!-----
    print*, 'DBG: maketiles -- re-normalizing tiles', ' (',iam,')
    do r=1,lnr
        do c=1,lis%d%gnc
            rsum=0.0
            do t=1,lis%p%nt
                if(fgrd(c,r,t).lt.lis%d%mina)then
                    fgrd(c,r,t)=0.0
                endif
                rsum=rsum+fgrd(c,r,t)
            enddo
            if(rsum.gt.0.0) then
                do t=1,lis%p%nt
                    if(rsum.gt.0.0)fgrd(c,r,t)=fgrd(c,r,t)/rsum
                enddo

                rsum=0.0
                do t=1,lis%p%nt
                    rsum=rsum+fgrd(c,r,t)
                enddo

                if(rsum.lt.0.9999.or.rsum.gt.1.0001)then
                    write(*,*) 'error1 in vegetation tiles',rsum,c,r
                endif
            endif
        enddo
    enddo
!-----
! Exclude tiles with MAXT (Maximum Tiles per grid),
! normalize remaining tiles to 100%
! Determine the grid predominance order of the tiles
! PVEG(NT) will contain the predominance order of tiles
!-----
    print*, 'DBG: maketiles -- excluding MAXT tiles', ' (',iam,')
    do r=1,lnr
        do c=1,lis%d%gnc
            do t=1,lis%p%nt
                fvt(t)=fgrd(c,r,t)
                pveg(c,r,t)=0
            enddo
        enddo
    enddo

```

```

enddo
do i=1,lis%p%nt
  max=0.0
  t=0
  do j=1,lis%p%nt
    if(fvt(j).gt.max)then
      if(fgrd(c,r,j).gt.0) then
        max=fvt(j)
        t=j
      endif
    endif
  enddo
  if(t.gt.0) then
    pveg(c,r,t)=i
    fvt(t)=-999.0
  endif
enddo
enddo
!-----
! Impose MAXT Cutoff
!-----
print*,'DBG: maketiles -- imposing MAXT cut-off',',(,iam,,')

do r=1,lnr
  do c=1,lis%d%gnc
    rsum=0.0
    do t=1,lis%p%nt
      if(pveg(c,r,t).lt.1) then
        fgrd(c,r,t)=0.0
        pveg(c,r,t)=0
      endif
      if(pveg(c,r,t).gt.lis%d%maxt) then
        fgrd(c,r,t)=0.0          ! impose maxt cutoff
        pveg(c,r,t)=0
      endif
      rsum=rsum+fgrd(c,r,t)
    enddo
  enddo
!-----
! Renormalize veg fractions within a grid to 1
!-----
if(rsum.gt.0.0) then
  do t=1,lis%p%nt
    if(rsum.gt.0.0)fgrd(c,r,t)= fgrd(c,r,t)/rsum
  enddo

  rsum=0.0
  do t=1,lis%p%nt

```

```

        rsum=rsum+ fgrd(c,r,t) !recalculate rsum to check
      enddo
      tsum(c,r)=rsum

      if(rsum.lt.0.9999.or.rsum.gt.1.0001)then !check renormalization
        write(*,*) 'error2 in vegetation tiles',rsum,c,r
      endif
      endif

      enddo
    enddo
    deallocate(pveg)
    call absoft_release_cache()

if ( masterproc ) then
  allocate(glbmask(lis%d%gnc, lis%d%gnr), stat=ierr)
else
  allocate(glbmask(1, 1), stat=ierr)
endif
call check_error(ierr,'Error allocating glbmask.',iam)
if ( npes > 1 ) then
  call MPI_GATHERV(mask,lis%d%gnc*lnr,MPI_REAL, &
    glbmask,recvcnts2,displs2,MPI_REAL, &
    0,MPI_COMM_WORLD, ierr)
else
  glbmask = mask
endif
deallocate(mask)
call absoft_release_cache()

if ( masterproc ) then
  allocate(glblat(lis%d%gnc, lis%d%gnr), stat=ierr)
else
  allocate(glblat(1, 1), stat=ierr)
endif
call check_error(ierr,'Error allocating glblat.',iam)
if ( npes > 1 ) then
  call MPI_GATHERV(lat,lis%d%gnc*lnr,MPI_REAL, &
    glblat,recvcnts2,displs2,MPI_REAL, &
    0,MPI_COMM_WORLD, ierr)
else
  glblat = lat
endif
deallocate(lat)
call absoft_release_cache()

if ( masterproc ) then
  allocate(gblon(lis%d%gnc, lis%d%gnr), stat=ierr)

```

```

else
    allocate(glblon(1, 1), stat=ierr)
endif
call check_error(ierr,'Error allocating glblon.',iam)
if ( npes > 1 ) then
    call MPI_GATHERV(lon,lis%d%gnc*lnr,MPI_REAL, &
                      glblon,recvcnts2,displs2,MPI_REAL, &
                      0,MPI_COMM_WORLD, ierr)
else
    glblon = lon
endif
deallocate(lon)
call absoft_release_cache()

if ( masterproc ) then
    allocate(glbfgrd(lis%d%gnc, lis%d%gnr, lis%p%nt), stat=ierr)
else
    allocate(glbfgrd(1, 1, 13), stat=ierr)
endif
call check_error(ierr,'Error allocating glbfgrd.',iam)
if ( npes > 1 ) then
    do t = 1, lis%p%nt
        call MPI_GATHERV(fgrd(:,:,t),lis%d%gnc*lnr,MPI_REAL, &
                          glbfgrd(:,:,t),recvcnts2,displs2,MPI_REAL, &
                          0,MPI_COMM_WORLD, ierr)
    enddo
else
    glbfgrd = fgrd
endif
deallocate(fgrd)
call absoft_release_cache()

if ( masterproc ) then
    allocate(glbtsum(lis%d%gnc, lis%d%gnr), stat=ierr)
else
    allocate(glbtsum(1, 1), stat=ierr)
endif
call check_error(ierr,'Error allocating glbtsum.',iam)
if ( npes > 1 ) then
    call MPI_GATHERV(tsum,lis%d%gnc*lnr,MPI_REAL, &
                      glbtsum,recvcnts2,displs2,MPI_REAL, &
                      0,MPI_COMM_WORLD, ierr)
else
    glbtsum = tsum
endif
deallocate(tsum)
call absoft_release_cache()

```

```

if ( masterproc ) then
  landnveg = 5
!  if (lis%p%koster .eq. 1) landnveg = 4
  lis%d%glbnch=0
  do t=1,lis%p%nt
    do r=1,lis%d%gnr
      do c=1,lis%d%gnc
        if(glbmask(c,r).gt.0.99.and. &
           glbmask(c,r).lt.3.01)then
          if(glbfgrd(c,r,t).gt.0.0)then
            lis%d%glbnch=lis%d%glbnch+1
          endif
          if(glbtsum(c,r).eq.0.0.and.t.eq.landnveg)then
            lis%d%glbnch=lis%d%glbnch+1
          endif
        endif
      enddo
    enddo
  enddo

print*, 'DBG: maketiles -- glbnch',lis%d%glbnch,' (',iam,')'
allocate(tile(lis%d%glbnch))

lis%d%glbngrid=0
do r=1,lis%d%gnr
  do c=1,lis%d%gnc
    if(glbmask(c,r).gt.0.99 .and. &
       glbmask(c,r).lt.3.01) then
      lis%d%glbngrid=lis%d%glbngrid+1
    endif
  enddo
enddo
count = 1
print*, 'DBG: maketiles1 -- glbnch',lis%d%glbnch,' (',iam,')'
allocate(grid(lis%d%glbngrid))
allocate(glbgindex(lis%d%gnc, lis%d%gnr))
print*, 'DBG: maketiles2 -- glbnch',lis%d%glbnch,' (',iam,')'
do r=1,lis%d%gnr
  do c=1,lis%d%gnc
    glbgindex(c,r) = -1
    if(glbmask(c,r).gt.0.99 .and. &
       glbmask(c,r).lt.3.01) then
      grid(count)%lat = glblat(c,r)
      grid(count)%lon = glblon(c,r)
      grid(count)%fgrd = glbfgrd(c,r,:)
      glbgindex(c,r) = count
      count = count+1
    endif
  enddo
enddo

```

```

        enddo
    enddo
    deallocate(glblat,stat=ierr)
    call absoft_release_cache()
    call check_error(ierr,'Error allocating glblat.',iam)
    deallocate(glblon, stat=ierr)
    call check_error(ierr,'Error allocating glblon.',iam)
    print*, 'DBG: maketiles3 -- glbnch',lis%d%glbnch,' (',iam,')
    count = 0
    do r=1,lis%d%gnr
        do c=1,lis%d%gnc
            do t=1,lis%p%nt
                if(glbmask(c,r).gt.0.99.and. &
                   glbmask(c,r).lt.3.01)then
                    if(glbfgrd(c,r,t).gt.0.0)then
                        count = count+1
                        tile(count)%row=r
                        tile(count)%col=c
                        tile(count)%index = glbgindex(c,r)
                        tile(count)%vegt=t
                        tile(count)%fgrd=glbfgrd(c,r,t)
                    endif
                    if(glbtsum(c,r).eq.0.0.and.t.eq.landnveg)then
                        count=count+1
                        tile(count)%row=r
                        tile(count)%col=c
                        tile(count)%index = glbgindex(c,r)
                        tile(count)%vegt=t
                        tile(count)%fgrd=1.0
                    endif
                endif
            enddo
        enddo
    enddo
    print*, 'DBG: maketiles4 -- glbnch',lis%d%glbnch,' (',iam,')
    deallocate(glbmask, stat=ierr)
    call check_error(ierr,'Error allocating glbmask',iam)
    deallocate(glbfgrd, stat=ierr)
    call check_error(ierr,'Error allocating glbfgrd',iam)
    deallocate(glbtsum, stat=ierr)
    call check_error(ierr,'Error allocating glbtsum.',iam)
    call absoft_release_cache()
    WRITE(*,*) 'MSG: maketiles -- Size of Tile Dimension:',NCHP, &
               ' (',iam,')
    WRITE(*,*) 'MSG: maketiles -- Actual Number of Tiles:', &
               LIS%D%GLBNCH,' (',iam,')
    WRITE(*,*)
```

```

      WRITE(*,*) 'MSG: maketiles -- Size of Grid Dimension:', &
                  lis%d%glbngrid,' (',iam,')'
      WRITE(*,*) 
      WRITE(79,*), 'MSG: maketiles -- Size of Tile Dimension:',NCHP, &
                  ' (',iam,')'
      WRITE(79,*), 'MSG: maketiles -- Actual Number of Tiles:', &
                  LIS%D%GLBNCH,' (',iam,')'
      WRITE(79,*)

      endif
      print*, 'MSG: maketiles -- done', ' (',iam,')'

#endif
      return

```

1.9.19 maketiles_nongds.F90 (Source File: maketiles_nongds.F90)

This primary goal of this routine is to determine tile space for MPI-based I/O

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Major F90 and major structure revision
3 Jan 2000: Minor T=0 bug fix, should have no effect on output
8 Mar 2000: Brian Cosgrove; Initialized FGRD to 0 For Dec Alpha Runs
22 Aug 2000: Brian Cosgrove; Altered code for US/Mexico/Canada Mask
04 Feb 2001: Jon Gottschalck; Added option to read and use Koster tile space
17 Oct 2003: Sujay Kumar ; Initial version of subsetting code

```

INTERFACE:

```
subroutine maketiles_nongds()
```

USES:

```

use lisdrv_module, only: lis, grid, glbgindex, tile
use grid_module
use spmdMod

```

CONTENTS:

```

if ( masterproc ) then
  if(lis%d%kgds(42) > lis%d%lnc .or. &
     lis%d%kgds(43) > lis%d%lnr) then !using a subdomain
    gnc = lis%d%kgds(42)
    gnr = lis%d%kgds(43)
  else

```

```

gnc = lis%d%lnc
gnr = lis%d%lnr
endif
lis%d%gnc = gnc
lis%d%gnr = gnr

allocate(lat(gnc,gnr), stat=ierr)
call check_error(ierr,'Error allocating lat.',iam)

allocate(lon(gnc,gnr), stat=ierr)
call check_error(ierr,'Error allocating lon.',iam)

allocate(mask(gnc, gnr), stat=ierr)
call check_error(ierr,'Error allocating mask.',iam)

nchp = lis%d%glbnch

print*, 'MSG: maketiles -- Reading ',trim(lis%p%myfile), &
      ' (',iam,')'
open(30,file=lis%p%myfile,form='unformatted',status='old')
read(30) lat
read(30) lon
read(30) mask
close(30)
print*, 'MSG: maketiles -- Done reading ',trim(lis%p%myfile), &
      ' (',iam,')'
allocate(locallat(lis%d%lnc,lis%d%lnr))
allocate(locallon(lis%d%lnc,lis%d%lnr))
allocate(localmask(lis%d%lnc,lis%d%lnr))
localmask = 0

do r=1,gnr
  do c=1,gnc
    if(lat(c,r)*1000.ge.lis%d%kgds(4).and. &
       lat(c,r)*1000.le.lis%d%kgds(7).and. &
       lon(c,r)*1000.ge.lis%d%kgds(5).and. &
       lon(c,r)*1000.le.lis%d%kgds(8)) then
      rindex = r - (lis%d%kgds(4)-lis%d%kgds(44)) &
                 /lis%d%kgds(9)
      cindex = c - (lis%d%kgds(5)-lis%d%kgds(45)) &
                 /lis%d%kgds(10)
      locallat(cindex,rindex) = lat(c,r)
      locallon(cindex,rindex) = lon(c,r)
      localmask(cindex,rindex) = mask(c,r)
    endif
  end do
end do
deallocate(mask)

```

```

call absoft_release_cache()

if(lis%f%ecor .eq. 1) then
    allocate(tmpelev(gnc*gnr),stat=ierr)
    call check_error(ierr,'Error allocating elevation.',iam)

    allocate(elevdiff(lis%d%lnc,lis%d%lnr), stat=ierr)
    call check_error(ierr,'Error allocating elev diff.',iam)

    elevdiff = 0.0
    tmpelev = 0.0
    open(21,file=lis%p%elevfile,form='unformatted',&
          status='old',iostat=ierr)
    if (ierr /= 0) then
        write(79,*) "stop: problem opening elevation difference file"
        write(79,*) "try running without elevation correction option."
        print*, "stop: problem opening elevation difference file"
        print*, "try running without elevation correction option."
        call endrun
    else
        read (21) (tmpelev(ppp), ppp = 1,gnc*gnr)
    endif
    close(21)

    cc=0
    do r=1,lis%d%gnr
        do c=1,lis%d%gnc
            if(lat(c,r)*1000.ge.lis%d%kgds(4).and. &
               lat(c,r)*1000.le.lis%d%kgds(7).and. &
               lon(c,r)*1000.ge.lis%d%kgds(5).and. &
               lon(c,r)*1000.le.lis%d%kgds(8)) then
                rindex = r - (lis%d%kgds(4)-lis%d%kgds(44)) &
                           /lis%d%kgds(9)
                cindex = c - (lis%d%kgds(5)-lis%d%kgds(45)) &
                           /lis%d%kgds(10)
                elevdiff(cindex,rindex) = tmpelev(c+cc)
            endif
        enddo
        cc = cc + lis%d%gnc
    enddo
    print*, 'done reading elevation difference file..'
    deallocate(tmpelev)
    call absoft_release_cache()
endif

allocate(fgrd(lis%d%lnc,lis%d%lnr,lis%p%nt), stat=ierr)
call check_error(ierr,'Error allocating fgrd.',iam)

```

```

allocate(veg(gnc, gnr,lis%p%nt), stat=ierr)
call check_error(ierr,'Error allocating veg.',iam)

fgrd = 0
!-----
! Select which tile-space veg. info to use (UMD or Koster)
!-----
print*, 'MSG: maketiles -- Reading ',trim(lis%p%vfile), &
' (',iam,')'
open(98,file=lis%p%vfile,form='unformatted')
read(98) lat
read(98) lon
do t = 1, lis%p%nt
    read(98) veg(:,:,t)
enddo
print*, 'MSG: maketiles -- Done reading ',trim(lis%p%vfile), &
' (',iam,')'
do r=1,gnr
    do c=1,gnc
        isum=0.0
        if(lat(c,r)*1000.ge.lis%d%kgds(4).and. &
           lat(c,r)*1000.le.lis%d%kgds(7).and. &
           lon(c,r)*1000.ge.lis%d%kgds(5).and. &
           lon(c,r)*1000.le.lis%d%kgds(8)) then
            rindex = r - (lis%d%kgds(4)-lis%d%kgds(44)) &
                      /lis%d%kgds(9)
            cindex = c - (lis%d%kgds(5)-lis%d%kgds(45)) &
                      /lis%d%kgds(10)
            do t=1,lis%p%nt
                isum=isum+veg(c,r,t) !recompute ISUM without water points
            enddo
            do t=1,lis%p%nt
                fgrd(cindex,rindex,t)=0.0
                if(isum.gt.0) fgrd(cindex,rindex,t)=veg(c,r,t)/isum
            enddo
        end if
    enddo
enddo
close(98)

print*, 'MSG: maketiles -- Done calculations with ',&
       trim(lis%p%vfile), &
       ' (',iam,')'
call absoft_release_cache()

allocate(tsum(lis%d%lnc, lis%d%lnr), stat=ierr)
call check_error(ierr,'Error allocating tsum.',iam)
tsum = 0.0

```

```

!-----
! Exclude tiles with MINA (minimum tile grid area),
! normalize remaining tiles to 100%
!-----

      do r=1,lis%d%lnr
        do c=1,lis%d%lnc
          rsum=0.0
          do t=1,lis%p%nt
            if(fgrd(c,r,t).lt.lis%d%mina)then
              fgrd(c,r,t)=0.0
            endif
            rsum=rsum+fgrd(c,r,t)
          enddo
        !-----
        ! renormalize veg fractions within a grid to 1
        !-----
        if(rsum.gt.0.0) then
          do t=1,lis%p%nt
            if(rsum.gt.0.0)fgrd(c,r,t)=fgrd(c,r,t)/rsum
          enddo

          rsum=0.0
          do t=1,lis%p%nt
            rsum=rsum+fgrd(c,r,t)
          enddo

          if(rsum.lt.0.9999.or.rsum.gt.1.0001)then
            write(*,*) 'Error1 in vegetation tiles',rsum,c,r
          endif
        endif
        enddo
      enddo

      allocate(pveg(lis%d%lnc,lis%d%lnr,lis%p%nt), stat=ierr)
      call check_error(ierr,'Error allocating pveg.',iam)

!-----
! Exclude tiles with MAXT (Maximum Tiles per grid),
!   normalize remaining tiles to 100%
! Determine the grid predominance order of the tiles
!   PVEG(NT) will contain the predominance order of tiles
!-----

      do r=1,lis%d%lnr
        do c=1,lis%d%lnc
          do t=1,lis%p%nt
            fvt(t)=fgrd(c,r,t)
            pveg(c,r,t)=0
          enddo
          do i=1,lis%p%nt

```

```

max=0.0
t=0
do j=1,lis%p%nt
  if(fvt(j).gt.max)then
    if(fgrd(c,r,j).gt.0) then
      max=fvt(j)
      t=j
    endif
  endif
enddo
if(t.gt.0) then
  pveg(c,r,t)=i
  fvt(t)=-999.0
endif
enddo
enddo
!-----
! Impose MAXT Cutoff
!-----
do r=1,lis%d%lnr
  do c=1,lis%d%lnc
    rsum=0.0
    do t=1,lis%p%nt
      if(pveg(c,r,t).lt.1) then
        fgrd(c,r,t)=0.0
        pveg(c,r,t)=0
      endif
      if(pveg(c,r,t).gt.lis%d%maxt) then
        fgrd(c,r,t)=0.0
        pveg(c,r,t)=0
      endif
      rsum=rsum+fgrd(c,r,t)
    enddo
  enddo
!-----
! renormalize veg fractions within a grid to 1
!-----
if(rsum.gt.0.0) then
  do t=1,lis%p%nt
    if(rsum.gt.0.0)fgrd(c,r,t)= fgrd(c,r,t)/rsum
  enddo

  rsum=0.0
  do t=1,lis%p%nt
    rsum=rsum+ fgrd(c,r,t) !recalculate rsum to check
  enddo
  tsum(c,r)=rsum

```

```

        if(rsum.lt.0.9999.or.rsum.gt.1.0001)then !check renormalization
            write(*,*) 'Error2 in vegetation tiles',rsum,c,r
        endif
    endif
enddo
deallocate(pveg)
call absoft_release_cache()

landnveg = 5
!-----
! Make Tile Space
!-----
lis%d%glbnch=0
do t=1,lis%p%nt
    do r=1,lis%d%lnr
        do c=1,lis%d%lnc
            if(localmask(c,r).gt.0.99.and. &
               localmask(c,r).lt.3.01)then !we have land
                if(fgrd(c,r,t).gt.0.0)then
                    lis%d%glbnch=lis%d%glbnch+1
                endif
                if(tsum(c,r).eq.0.0.and.t.eq.landnveg)then
                    lis%d%glbnch=lis%d%glbnch+1
                endif
            endif
        enddo
    enddo
enddo

print*, 'DBG: maketiles -- glbnch',lis%d%glbnch,' (,iam,)'
allocate(tile(lis%d%glbnch))

lis%d%glbngrid=0
do r=1,lis%d%lnr
    do c=1,lis%d%lnc
        if(localmask(c,r).gt.0.99 .and. &
           localmask(c,r).lt.3.01) then
            lis%d%glbngrid=lis%d%glbngrid+1
        endif
    enddo
enddo
count = 1
print*, 'DBG: maketiles1 -- glbnch',lis%d%glbnch,' (,iam,)'
allocate(grid(lis%d%glbngrid))
allocate(glbindex(lis%d%lnc, lis%d%lnr))
print*, 'DBG: maketiles2 -- glbnch',lis%d%glbnch,' (,iam,)'
do r=1,lis%d%lnr

```

```

do c=1,lis%d%lnc
    glbgindex(c,r) = -1
    if(localmask(c,r).gt.0.99 .and. &
        localmask(c,r).lt.3.01) then
        grid(count)%lat = locallat(c,r)
        grid(count)%lon = locallon(c,r)
        grid(count)%fgrd = fgrd(c,r,:)
        glbgindex(c,r) = count
        count = count+1
    endif
enddo
enddo
deallocate(locallat,stat=ierr)
call absoft_release_cache()
call check_error(ierr,'Error allocating glblat.',iam)
deallocate(locallon, stat=ierr)
call check_error(ierr,'Error allocating glblon.',iam)
print*, 'DBG: maketiles3 -- glnch',lis%d%glnch,' (',iam,')'
!-----
!     For writing dominant Vegetation types
!-----
if(lis%o%wparam .eq.1) then
    allocate(domveg(lis%d%lnc,lis%d%lnr))
    domveg = 0
endif
count = 0
do r=1,gnr
    do c=1,gnc
        if(lat(c,r)*1000.ge.lis%d%kgds(4).and. &
            lat(c,r)*1000.le.lis%d%kgds(7).and. &
            lon(c,r)*1000.ge.lis%d%kgds(5).and. &
            lon(c,r)*1000.le.lis%d%kgds(8)) then
            rindex = r - (lis%d%kgds(4)-lis%d%kgds(44)) &
                /lis%d%kgds(9)
            cindex = c - (lis%d%kgds(5)-lis%d%kgds(45)) &
                /lis%d%kgds(10)
            do t=1,lis%p%nt
                if(localmask(cindex,rindex).gt.0.99.and. &
                    localmask(cindex,rindex).lt.3.01)then
                    if(fgrd(cindex,rindex,t).gt.0.0)then
                        count = count+1
                        tile(count)%row=r
                        tile(count)%col=c
                        tile(count)%index = glbgindex(cindex,rindex)
                        tile(count)%vegt=t
                        if(lis%o%wparam.eq.1) then
                            domveg(cindex,rindex) = t*1.0
                        endif
                    endif
                endif
            enddo
        endif
    enddo
enddo

```

```

        tile(count)%fgrd=fgrd(cindex,rindex,t)
        if(lis%f%ecor.eq.1) &
            tile(count)%elev=elevdiff(cindex,rindex)
        endif
!-----
! What if we have land without vegetation assigned
!-----
        if(tsum(cindex,rindex).eq.0.0.and.t.eq.landnveg)then
            count=count+1
            tile(count)%row=r
            tile(count)%col=c
            tile(count)%index = glbgindex(cindex,rindex)
            tile(count)%vegt=t
            if(lis%o%wparam.eq.1) then
                domveg(cindex,rindex) = t*1.0
            endif
            tile(count)%fgrd=1.0
            if(lis%f%ecor.eq.1) &
                tile(count)%elev=elevdiff(cindex,rindex)
            endif
            endif
        enddo
        endif
    enddo
    endif
enddo
if(lis%o%wparam.eq.1) then
    open(32,file="domvegtype.bin",form='unformatted')
    write(32) domveg
    close(32)
    deallocate(domveg)
endif
if(lis%f%ecor.eq.1)  deallocate(elevdiff)
deallocate(lat)
deallocate(lon)
print*, 'DBG: maketiles4 -- glbnch',lis%d%glbnch,' (',iam,')'
deallocate(localmask, stat=ierr)
call check_error(ierr,'Error allocating glbmask',iam)
deallocate(fgrd, stat=ierr)
call check_error(ierr,'Error allocating glbfgrd',iam)
deallocate(tsum, stat=ierr)
call check_error(ierr,'Error allocating glbtsum.',iam)
call absoft_release_cache()

WRITE(*,*) 'MSG: maketiles -- Size of Tile Dimension:',NCHP, &
' (',iam,')'
WRITE(*,*) 'MSG: maketiles -- Actual Number of Tiles:', &
LIS%D%GLBNCH, ' (',iam,')'
WRITE(*,*)
```

```

      WRITE(*,*) 'MSG: maketiles -- Size of Grid Dimension:', &
      lis%d%glbngrid,' (',iam,')'
      WRITE(*,*)

      WRITE(79,*) 'MSG: maketiles -- Size of Tile Dimension:',NCHP, &
      ' (',iam,')'
      WRITE(79,*) 'MSG: maketiles -- Actual Number of Tiles:', &
      LIS%D%GLBNCH,' (',iam,')'
      WRITE(79,*)
      endif
      print*, 'MSG: maketiles -- done', ' (',iam,')'
      return

```

1.10 Fortran: Module Interface mpishorthand.F90 (Source File: mpishorthand.F90)

Data and parameters used for MPI. Some shorthand variables with shorter names than the standard MPI parameters. Also some variables used for heap management. Adopted from CLM

REVISION HISTORY:

02 Jan 2002 : Sujay Kumar: Initial Version

INTERFACE:

```
module mpishorthand
```

USES:

```
#if (defined OSF1)
  include 'mpif.h'
#else
  use mpi
#endif
```

1.11 Fortran: Module Interface obsprecipforcing_module.F90 (Source File: obsprecipforcing_module.F90)

This module contains interfaces and subroutines that controls the incorporation of observed precipitation forcing

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

INTERFACE:

```
module obsprecipforcing_module
```

ARGUMENTS:

```
real, pointer :: obsprecip(:)
```

1.11.1 LIS_obsprecipforcing_init (Source File: *obsprecipforcing_module.F90*)**INTERFACE:**

```
interface LIS_obsprecipforcing_init
    module procedure precip_forcing_init
end interface
```

1.11.2 LIS_get_obsprecip_forcing (Source File: *obsprecipforcing_module.F90*)**INTERFACE:**

```
interface LIS_get_obsprecip_forcing
    module procedure get_obsprecip_forcing
end interface
```

1.11.3 precip_forcing_init (Source File: *obsprecipforcing_module.F90*)

Initializes the variables required for the spatial interpolation of observed precipitation data

INTERFACE:

```
subroutine precip_forcing_init()
```

USES:

```
use lisdrv_module, only :lis
use def_ipMod, only : allocate_ip, def_ip_input
use precipforcing_pluginMod, only : precipforcing_plugin
use spmdMod, only : masterproc
```

CONTENTS:

```
if(lis%f%gpcpsrc.gt.0) then
    call precipforcing_plugin
    if(masterproc) then
        call defnatrespcp(lis%f%gpcpsrc)
        allocate(obsprecip(lis%d%ngrid))
    endif
endif
```

1.11.4 get_precip_forcing (Source File: *obsprecipforcing_module.F90*)

Calls the routines to open, read, and interpolate observed precipitation forcing data

INTERFACE:

```
subroutine get_obsprecip_forcing
```

USES:

```
use lisdrv_module, only: lis, gindex
use grid_spmdMod
```

CONTENTS:

```
if(lis%f%gpcpsrc.gt.0) then
    if(masterproc) then
        call glbpPrecip(lis%f%gpcpsrc)
    endif
    if(lis%f%findtime1 ==1 .or. &
       lis%f%findtime2==1) then
        if(npes > 1) then
            call scatter_precip_data()
        endif
        call timeinterppcp(lis%f%gpcpsrc)
    endif
endif
```

1.11.5 scatter_precip_data (Source File: *obsprecipforcing_module.F90*)

Distributes the precipitation forcing data onto the compute nodes

INTERFACE:

```
subroutine scatter_precip_data()
```

USES:

```
use grid_spmdMod
```

1.12 Fortran: Module Interface *obsradforcing_module.F90* (Source File: *obsradforcing_module.F90*)

This module contains interfaces and subroutines that controls the incorporation of observed radiation forcing

REVISION HISTORY:

14Nov02 Sujay Kumar Initial Specification

INTERFACE:

```
module obsradforcing_module
  implicit none
```

ARGUMENTS:

```
real, pointer :: oswdata1(:)
real, pointer :: oswdata2(:)
real, pointer :: oblwdta1(:)
real, pointer :: oblwdta2(:)
integer :: sstat1, sstat2, lstat1,lstat2
```

1.12.1 LIS_obsradforcing_init (Source File: *obsradforcing_module.F90*)

INTERFACE:

```
interface LIS_obsradforcing_init
  module procedure rad_forcing_init
end interface
```

1.12.2 LIS_get_obsrad_forcing (Source File: *obsradforcing_module.F90*)

INTERFACE:

```
interface LIS_get_obsrad_forcing
  module procedure get_obsrad_forcing
end interface
```

1.12.3 rad_forcing_init (Source File: *obsradforcing_module.F90*)

Allocates memory for variables required for radiation forcing interpolation

INTERFACE:

```
subroutine rad_forcing_init()
```

USES:

```
use lisdrv_module, only:lis
use grid_spmdMod
use radforcing_pluginMod, only :radforcing_plugin
```

CONTENTS:

```

if(lis%f%radsrc.gt.0) then
  call radforcing_plugin
  if(masterproc) then
    call defnatresrad(lis%f%radsrc)
    allocate(obsodata1(lis%d%ngrid))
    allocate(obsodata2(lis%d%ngrid))
    allocate(oblwdata1(lis%d%ngrid))
    allocate(oblwdata2(lis%d%ngrid))
  else
    allocate(obsodata1(gdi(iam)))
    allocate(obsodata2(gdi(iam)))
    allocate(oblwdata1(gdi(iam)))
    allocate(oblwdata2(gdi(iam)))
  endif
endif

```

1.12.4 get_obsrad_forcing (Source File: obsradforcing_module.F90)

Calls the routines that read observed radiation forcing methods

INTERFACE:

```
subroutine get_obsrad_forcing
```

USES:

```

use lisdrv_module, only: lis, grid
use grid_spmdMod
use driverpardef_module

```

CONTENTS:

```

sstat1 = 0
sstat2 = 0
lstat1 = 0
lstat2 = 0
if(lis%f%radsrc.gt.0) then
  call MPI_GATHERV(grid(1:gdi(iam)),gdi(iam), &
    MPI_GRID_STRUCT,grid,gdi,gdisp,MPI_GRID_STRUCT, &
    0,MPI_COMM_WORLD, ier)
  if(masterproc) then
    call getrad(lis%f%radsrc)
  endif
  call MPI_BCAST(sstat1, 1,MPI_INTEGER,0, &
    MPI_COMM_WORLD, ier)
  call MPI_BCAST(sstat2, 1,MPI_INTEGER,0, &

```

```

        MPI_COMM_WORLD, ier)
call MPI_BCAST(lstat1, 1,MPI_INTEGER,0, &
               MPI_COMM_WORLD, ier)
call MPI_BCAST(lstat2, 1,MPI_INTEGER,0, &
               MPI_COMM_WORLD, ier)
call MPI_BCAST(lis%f%findagrtme1,1,MPI_INTEGER,0, &
               MPI_COMM_WORLD, ier)
call MPI_BCAST(lis%f%findagrtme2,1,MPI_INTEGER,0, &
               MPI_COMM_WORLD, ier)
if(lis%f%findagrtme1 ==1 .or. &
   lis%f%findagrtme2==1) then
  if(npes > 1) then
    call scatter_rad_data()
  endif
endif
call timeinterprad(lis%f%radsrc)
endif

```

1.12.5 scatter_rad_data (Source File: obsradforcing_module.F90)

Distributes radiation forcing data on to the compute node.

INTERFACE:

```
subroutine scatter_rad_data()
```

USES:

```
use grid_spmdMod
use lisdrv_module, only : lis
```

1.13 Fortran: Module Interface opendap_module.F90 (Source File: open-dap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O

REVISION HISTORY:

10 Jul 2003; James Geiger Initial Specification

INTERFACE:

```
module opendap_module
#if ( defined OPENDAP )
```

USES:

```

use ESMF_TimeMgmtMod
use time_manager
use lisdrv_module, only : lis, grid, gindex
use grid_spmdMod,   only : gdi, gdisp
use tile_spmdMod,   only : di_array
use spmdMod

implicit none

```

ARGUMENTS:

integer	:: tnroffset	!Global to local row mapping offset
integer	:: grid_offset	!Global to local grid mapping offset
integer	:: output_slat	!Southern latitude boundary for the interpolated domain
integer	:: output_nlat	!Northern latitude boundary for the interpolated domain
integer	:: parm_slat	!Southern latitude boundary for each subdomain (for parameter data)
integer	:: parm_nlat	!Northern latitude boundary for each subdomain (for parameter data)
integer	:: parm_wlon	!Western longitude boundary for each subdomain (for parameter data)
integer	:: parm_elon	!Eastern longitude boundary for each subdomain (for parameter data)
integer	:: parm_nc	!Number of columns for each subdomain (for parameter data)
integer	:: parm_nr	!Number of rows for each subdomain (for parameter data)
character*3	:: ciam	!character representation of processor id
character*4	:: cpParm_slat	!character representation of cpParm\$-\$slat
character*4	:: cpParm_nlat	!character representation of cpParm\$-\$nlat
character*4	:: cpParm_wlon	!character representation of cpParm\$-\$wlon
character*4	:: cpParm_elon	!character representation of cpParm\$-\$elon
integer	:: nroffset	

```

#if ( defined ABSOFT )
    character*15 :: opendap_home = '/home/jim/'
#else
    character*2  :: opendap_home = './'
#endif

```

1.13.1 opendap_init (Source File: opendap_module.F90)

Initializes the GDS variables

INTERFACE:

```
subroutine opendap_init()
```

CONTENTS:

```

call init_parm_vars()
call reset_lis_filepaths()

```

1.13.2 reset_lis_filepaths (Source File: *opendap_module.F90*)

Resets various input data filenames for execution through GDS

INTERFACE:

```
subroutine reset_lis_filepaths()
```

CONTENTS:

```
lis%p%clfile      = trim(opendap_home)//trim(adjustl(ciam))///'//lis%p%clfile
lis%p%safile      = trim(opendap_home)//trim(adjustl(ciam))///'//lis%p%safile
lis%p%iscfile     = trim(opendap_home)//trim(adjustl(ciam))///'//lis%p%iscfile
```

1.13.3 init_parm_vars (Source File: *opendap_module.F90*)

Computes domain decomposition for native as well as interpolated domains for input data

INTERFACE:

```
subroutine init_parm_vars()
```

CONTENTS:

```
select case (lis%d%domain)
case(1)
    print*, 'Error!  Cannot handle ndas.'
    stop 999
case(2)
    res = 250
    center = 875
    bottom = -59875
case(3)
    print*, 'Error!  Cannot handle 2x2.5.'
    stop 999
case(4)
    res = 1000
    center = 500
    bottom = -59500
case(5)
    res = 500
    center = 750
    bottom = -59750
case(6)
    res = 50
    center = 975
    bottom = -59975
case DEFAULT
    print*, "Select domain size (1,2,3,4,5,6)"
```

```

    stop 999
end select

call set_parm_lat(parm_slat,parm_nlat,output_slat,output_nlat,res,bottom)
parm_wlon = 1
parm_elon = lis%d%gnc

lis%d%ngrid = gdi(iam)
lis%d%nch   = di_array(iam)

parm_nc      = lis%d%gnc
parm_nr      = ( parm_nlat - parm_slat + 1 )

tnroffset   = parm_slat - 1
nroffset    = 0
grid_offset = tile(1)%index-1

write(ciam, '(i3)') iam
write(cparm_slat, '(i4)') parm_slat
write(cparm_nlat, '(i4)') parm_nlat
write(cparm_wlon, '(i4)') parm_wlon
write(cparm_elon, '(i4)') parm_elon

print*, 'DBG: parm_init -- parm_slat', parm_slat, ', iam, ')
print*, 'DBG: parm_init -- parm_nlat', parm_nlat, ', iam, ')
print*, 'DBG: parm_init -- ngid', lis%d%ngrid, ', iam, ')
print*, 'DBG: parm_init -- glbngrid', lis%d%glbngrid, ', iam, ')
print*, 'DBG: parm_init -- lnc', lis%d%lnc, ', iam, ')
print*, 'DBG: parm_init -- lnr', lis%d%lnr, ', iam, ')
print*, 'DBG: parm_init -- nroffset', nroffset, ', iam, ')
print*, 'DBG: parm_init -- tnroffset', tnroffset, ', iam, ')
print*, 'DBG: parm_init -- grid_offset', grid_offset, ', iam, ')
print*, 'DBG: parm_init -- cparm_slat', cparm_slat, ', iam, ')
print*, 'DBG: parm_init -- cparm_nlat', cparm_nlat, ', iam, ')
print*, 'DBG: parm_init -- ciam', ciam, ', iam, ')
print*, 'DBG: parm_init -- parm_nc', parm_nc, ', iam, ')
print*, 'DBG: parm_init -- parm_nr', parm_nr, ', iam, ')
line = '"/ciam//"'
```

1.13.4 set_parm_lat (Source File: opendap_module.F90)

Computes the latitudes of the decomposed domain for the parameter data

INTERFACE:

```
subroutine set_parm_lat(slat, nlat, oslat, onlat, res, bottom)
```

USES:

```
use lisdrv_module, only : tile
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: res, bottom
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: slat, nlat, oslat, onlat
```

CONTENTS:

```
!-----
! Set southern latitude index
!-----
slat = tile( 1 )%row
!-----
! Set southern latitude boundary
!-----
oslat = bottom + (slat-1)*res
!-----
! Set northern latitude index
!-----
nlat = tile( gdi(iam) )%row
!-----
! Set northern latitude boundary
!-----
onlat = bottom + (nlat-1)*res
end subroutine set_parm_lat
#endif
```

1.14 Fortran: Module Interface precision.F90 (Source File: precision.F90)

Define the precision to use for floating point and integer operations throughout the model.

REVISION HISTORY:

14 Nov 2002; Sujay Kumar Initial Specification

INTERFACE:

```
module precision
```

ARGUMENTS:

```
integer, parameter :: r4 = selected_real_kind(5)
integer, parameter :: r8 = selected_real_kind(6)
integer, parameter :: i8 = selected_int_kind(13)
```

1.14.1 readcard.F90 (Source File: readcard.F90)

Reads in LIS run specifics from lis.crd

REVISION HISTORY:

REVISION HISTORY:

- 15 Oct 1999: Paul Houser; Initial code
- 4 Apr 2000: Jeffrey Walker; Added catchment model output interval
- 11 Apr 2000: Brian Cosgrove; Added Elevation correction and Forcing Mask read statements
- 6 Jun 2000: Jon Radakovich; Updated for new version of CLM
- 23 Feb 2001: Urszula Jambor; Added GEOS or GDAS forcing option
- 27 Mar 2001: Jon Gottschalck; Revision of subroutine by implementing namelists
- 05 Sep 2001: Brian Cosgrove; Altered forcing logfile output to include more precip types
- 04 Feb 2002: Jon Gottschalck; Added section to set to Koster tilespace files if necessary
- 15 Apr 2002: Urszula Jambor; Added ECMWF forcing options, also adding 1 & 1/2 degree GLDAS domain options.
- 28 Apr 2002: Kristi Arsenault; Added NOAH LSM code
- 14 Nov 2003: Sujay Kumar; Modified card file that includes regional modeling options

INTERFACE:

```
subroutine readcard(lis)
```

USES:

```
use lis_module
implicit none
```

ARGUMENTS:

```
type (lisdec):: lis
```

CONTENTS:

```
open(10,file='lis.crd',form='formatted',status='old')
!-----
! Reading in parameters that need to be initialized
! to avoid any problems later
! Reading in parameters that are used by all LDAS runs
!-----
read(unit=10,nml=driver)
read(unit=10,nml=lis_run_inputs)
!-----
! Open runtime diagnostics file
!-----
call openfile(name,lis%o%odir,lis%o%expcode,lis%o%dfile)
88 format(a4,25x,a3,5x,16a)
89 format(20x,a49)
```

```

!-----
! Impose limit on time step
!-----
call check_timestep(lis%t%ts)
!-----
! Set Time
!-----
call set_time(lis%t)

call set_output_counters(lis%o%numoutf)

call date2time(lis%t%time,lis%t%doy,lis%t%gmt, &
    lis%t%yr,lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)

print*
print*, '***** GSFC-LIS driver *****'
print*, 'experiment code: ', '-' ,lis%o%expcode, '-'
print*, 'starting time: ',lis%t%smo,'/',lis%t%sda,'/',lis%t%syr
print*, 'ending time: ',lis%t%emo,'/',lis%t%eda,'/',lis%t%eyr
print*
read(unit=10,NML=domain)
print*, 'DOMAIN details: '
!-----
! Read namelist of parameters depending on the domain
!-----
lis%d%lnc = lis%d%kgds(2)
lis%d%lnr = lis%d%kgds(3)
print*, 'running domain', '-' ,lis%d%lnc,lis%d%lnr, '-'
print*, 'minimum tile area: ',lis%d%mina
print*, 'maximum tiles per grid: ',lis%d%maxt
read(unit=10,nml=parms)
print*, 'forcing details: '
!-----
! Setting Satellite LAI variables
!-----
lis%p%laitime = 0.0
if(lis%p%lai.eq.2) then
    print*, "Using AVHRR Satellite LAI"
endif
if(lis%p%lai.eq.3) then
    print*, "Using MODIS Satellite LAI"
endif

lis%f%rstflag = 1
lis%f%gridchange = 1
!-----
! Initialize Statistics files conditional
!-----
```

```

lis%o%foropen=0
!-----
! Fix MAXT out of bounds problems
!-----
if(lis%d%maxt.gt.lis%p%nt) lis%d%maxt=lis%p%nt
if(lis%d%maxt.lt.1      ) lis%d%maxt=1
!-----
! Select which vegetation tile space and mask files
!-----
print*, 'miscellaneous details:'
if(lis%p%vclass.eq.1) print*, 'avhrr umd vegetation', lis%p%vclass
if(lis%p%vclass.eq.2) print*, 'modis umd vegetation', lis%p%vclass
print*, 'mask file: ', lis%p%myfile
print*, 'vegetation file: ', lis%p%vfile
if (lis%d%soil.eq.1) print *, 'original vegetation-based soil scheme'
if (lis%d%soil.eq.2) print *, 'reynolds soils'
if (lis%d%soil.eq.1) print *, 'original vegetation-based soil scheme'
if (lis%d%soil.eq.2) print *, 'reynolds soils'

close(10)
return

```

1.14.2 check_timestep (Source File: readcard.F90)

check timestep to make sure it is between 1 and 3600

INTERFACE:

```
subroutine check_timestep(timestep)
```

CONTENTS:

```

if (timestep.gt.3600) then
  print *,'-----',
  print *,'error, user timestep > 3600 seconds!!!'
  print *,'resetting lis%ts to 3600!!!!'
  print *,'-----',
  timestep=3600
endif
if(timestep.lt.1) then
  print*,'Timestep can not be less than 1 minute, reset to 15 min.'
  timestep=15*60
endif
return

```

1.14.3 openfile (Source File: readcard.F90)

This subroutine puts together a filename

REVISION HISTORY:

4 Jan 2000: Paul Houser; Original Code

INTERFACE:

```
subroutine openfile(name,odir, expcode, ffile)
  implicit none
```

OUTPUT PARAMETERS:

```
character*80 name
```

INPUT PARAMETERS:

```
character*80 mkdir
character*40 odir,ffile
integer :: expcode
```

CONTENTS:

```
!-----
! Put together filename
!-----
92 format(80a1)
93 format(a80)
94 format(a4,i3,a1)
96 format(a40)
100 format(a9)

write(unit=temp,fmt='(a40)') odir
read(unit=temp,fmt='(80a1)') (fbase(i),i=1,80)
c=0
do i=1,80
  if(fbase(i).eq.( ' ').and.c.eq.0)c=i-1
enddo

write(unit=temp,fmt='(a4,i3,a1)')'/EXP',expcode,'/
read(unit=temp,fmt='(80a1)')(fcode(i),i=1,80)
d=0
do i=1,80
  if(fcode(i).eq.( ' ').and.d.eq.0)d=i-1
enddo
write(unit=temp,fmt='(a40)') ffile
read(unit=temp,fmt='(80a1)')(fname(i),i=1,80)
e=0
do i=1,80
  if(fname(i).eq.( ' ').and.e.eq.0)e=i-1
```

```

enddo

write(unit=temp,fmt='(80a1)')(fbase(i),i=1,c), &
    (fcode(i),i=1,d), (fname(i),i=1,e)
read(unit=temp,fmt='(a80)') name
write(unit=temp,fmt='(a9)')'mkdir -p '
read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)

!-----
! Make the directories for the output files
!-----

write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9), &
    (fbase(i),i=1,c),(fcode(i),i=1,d)
read(unit=temp,fmt='(a80)')mkdir
call system(mkdir)
return

```

1.15 Fortran: Module Interface spmdMod.F90 (Source File: spmdMod.F90)

MPI routines for initialization and computing arguments for different operations.

INTERFACE:

```
module spmdMod
```

ARGUMENTS:

```

#if (!defined SPMD)
    logical :: masterproc = .true. ! proc 0 logical for printing msgs
    integer :: iam = 0
#endif

#if (defined SPMD)

#if (defined OFFLINE)
    use mpishorthand
#endif

#if (defined OFFLINE)
    integer :: npes      !number of processors
    integer :: iam       !proc number
    logical :: masterproc !proc 0 logical for printing msgs
#endif

```

1.15.1 spmd_init (Source File: spmdMod.F90)

MPI initialization (number of cpus, processes, tids, etc)

INTERFACE:

```
subroutine spmd_init
DIR$ NAME (release_cache="_f90a_free_all")
```

CONTENTS:

```
#if (defined OFFLINE)
    call mpi_init(ier)
#endif
    call mpi_comm_rank(MPI_COMM_WORLD, iam, ier)
    if (iam==0) then
        masterproc = .true.
    else
        masterproc = .false.
    end if
    call mpi_comm_size(MPI_COMM_WORLD, npes, ier)
    return
```

1.15.2 stats.F90 (Source File: stats.F90)

Calculates statistics for a given variable

REVISION HISTORY:

Nov 11 1999: Jon Radakovich; Initial code

INTERFACE:

```
subroutine stats(var,udef,nch,mean,stdev,min,max)
```

ARGUMENTS:

```
integer :: nch
real :: var(nch)
real :: mean,dev,stdev,min,max,udef,vsum
```

CONTENTS:

```
vsum=0.
mean=0.
dev=0.
stdev=0.
min=100000.
max=-100000.
```

```

do t=1,nch
  if(var(t).ne.Undef)then
    vsum=vsum+var(t)
    if(var(t).gt.max)max=var(t)
    if(var(t).lt.min)min=var(t)
  endif
enddo
if(vsum.eq.0.)then
  max=0.
  min=0.
endif

mean=vsum/float(nch)

do t=1,nch
  if(var(t).ne.Undef)then
    dev=dev+(var(t)-mean)**2
  endif
enddo
stdev=(dev*(float(nch)-1)**(-1))**0.5

return

```

1.16 Fortran: Module Interface string_utils (Source File: string_utils.F90)

This module contains routines that perform string manipulations

REVISION HISTORY:

14 Nov 2002: Sujay Kumar: Initial Version, adopted from CLM

INTERFACE:

```
module string_utils
```

1.16.1 to_upper (Source File: string_utils.F90)

Convert character string to upper case.

Method: Use achar and iachar intrinsics to ensure use of ascii collating sequence.

INTERFACE:

```
function to_upper(str)
  implicit none
```

ARGUMENTS:

```
character(len=*), intent(in) :: str      ! String to convert to upper case
character(len=len(str))      :: to_upper
```

CONTENTS:

```
do i = 1, len(str)
    ctmp = str(i:i)
    aseq = iachar(ctmp)
    if ( aseq >= 97 .and. aseq <= 122 ) ctmp = achar(aseq - 32)
    to_upper(i:i) = ctmp
end do
```

1.16.2 t2gr.F90 (Source File: t2gr.F90)

Aggregate variables for all tiles at each grid point

REVISION HISTORY:

15 Oct 1999: Paul Houser; Initial Code

INTERFACE:

```
subroutine t2gr(t,g,ngrid, nch,tile)
```

USES:

```
use tile_module
```

CONTENTS:

```
g=0.0
do i=1,nch
    g(tile(i)%index)=g(tile(i)%index)+t(i)*tile(i)%fgrd
enddo
return
```

1.16.3 tile2grid.F90 (Source File: tile2grid.F90)

Transfer variables from tile space to grid space

REVISION HISTORY:

20 Oct 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine tile2grid(t,g,nch,nc,nr,tile)
```

USES:

```

use tile_module
use lisdrv_module, only :glbgindex

CONTENTS:
```

```

g=0.0
do r = 1, nr
  do c = 1, nc
    if(glbgindex(c,r).ne.-1) then
      g(c,r) = g(c,r)+t(glbgindex(c,r))*tile(glbgindex(c,r))%fgrd
    else
      g(c,r) = 9.999E20
    endif
  enddo
end do
return
```

1.16.4 tile_module.F90 (Source File: tile_module.F90)

LIS non-model-specific tile variables only.

REVISION HISTORY:

```

15 Oct 1999: Paul Houser; Initial code
22 Aug 2000: Brian Cosgrove; Modified code for output of
              standard LDAS output variables--added CC and AC,
              the canopy and aerodynamic conductance
```

INTERFACE:

```

module tile_module

  implicit none
  public tiledec
```

ARGUMENTS:

```

type tiledec
  integer :: col          !Grid Column of Tile
  integer :: row          !Grid Row of Tile
  integer :: index         !Index of corresponding grid
  integer :: vegt          !Vegetation Type of Tile
  real    :: fgrd          !Fraction of Grid covered by tile
  real    :: elev
end type tiledec
```

1.17 Fortran: Module Interface tile_spmdMod.F90 (Source File: tile_spmdMod.F90)

This module contains routines for domain decomposition in tile space

REVISION HISTORY:

14 Nov 2002; Sujay Kumar Initial Specification

INTERFACE:

```
module tile_spmdMod
```

USES:

```
use spmdMod
```

ARGUMENTS:

```
integer, allocatable :: di_array(:) !array containing the sizes of the decomposed tile space
integer, allocatable :: displs(:) !array containing relative displacements fo the tile space
```

1.17.1 allocate_tiled() (Source File: tile_spmdMod.F90)

Allocates memory for arrays containing tile decomposition information

INTERFACE:

```
subroutine allocate_tiled()
```

1.17.2 tile_spmd_init (Source File: tile_spmdMod.F90)

Performs domain decomposition in tile space

INTERFACE:

```
subroutine tile_spmd_init(tile, nch, nmif)
```

USES:

```
use tile_module
```

ARGUMENTS:

```
type(tileddec) :: tile(nch)
integer :: nch, nmif
```

CONTENTS:

```

ntiles = 1
deltax = nch/npes
do p=0,npes-2
    nti(p) = ntiles
    tindex = ntiles+deltax-1
    gind = tile(tindex)%index
    do while(tile(tindex+1)%index ==gind)
        tindex = tindex+1
    enddo
    ntf(p) = tindex
    ntiles = tindex+1
enddo
nti(npes-1) = ntiles
ntf(npes-1) = nch
do i=0,npes-1
    di_array(i) = ntf(i)-nti(i)+1
enddo

displs(0) = 0
do i = 1, npes-1
    displs(i) = displs(i-1)+di_array(i-1)
enddo

```

1.18 Fortran: Module Interface time_manager.F90 (Source File: time_manager.F90)

This module contains wrapper functions that uses the ESMF time manager for time management

INTERFACE:

```

module time_manager

#if (defined OFFLINE)

```

USES:

```

use precision, only: r8
use ESMF_TimeMgmtMod, only: &
    esmf_errhandlersettype, esmf_err_return, esmf_errprint, esmf_success, &
    esmf_time, esmf_timeinit, esmf_timeget, esmf_timegetdays, &
    esmf_timeincrement, esmf_timedecrement, &
    esmf_date, esmf_dateinit, esmf_gregorian, esmf_no_leap, esmf_dateget, &
    esmf_dateincrementsec, esmf_dateincrementday, esmf_datedecrement, &
    esmf_datediff, esmf_dategetfltdayofyear, &
    esmf_timemgr, esmf_timemgrinit, esmf_timemgradvance, esmf_timemgrgetnstep, &
    esmf_timemgrgetstepsize, esmf_timemgrgetstartdate, esmf_timemgrgetbasedate, &
    esmf_timemgrlaststep, esmf_timemgrgetcurrdate, esmf_timemgrgetprevdate, esmf_dateisla

```

```

    esmf_tiemmgrrestartwrite, esmf_tiemngrrestartread
    use string_utils, only: to_upper
#ifndef (defined SPMD)
    use spmdMod, only: mpicom, mpiint, masterproc
#else
    use spmdMod, only: masterproc
#endif

```

ARGUMENTS:

```

public :: &
    timemgr_init,                      &! time manager initialization
    advance_timestep,                  &! increment timestep number
    get_step_size,                     &! return step size in seconds
    get_nstep,                         &! return timestep number
    get_curr_date,                     &! return date components at end of current timestep
    get_prev_date,                     &! return date components at beginning of current timestep
    get_start_date,                   &! return date components of the start date
    get_ref_date,                      &! return date components of the reference date
    get_curr_time,                     &! return components of elapsed time since reference date
    get_curr_calday,                  &! return calendar day at end of current timestep
    is_first_step,                     &! return true on first step of initial run
    is_first_restart_step,             &! return true on first step of restart or branch run
    is_end_curr_day,                  &! return true on last timestep in current day
    is_end_curr_month,                &! return true on last timestep in current month
    is_last_step,                     &! return true on last timestep
    timemgr_write_restart,            &! write info to file needed to restart the time manager
    timemgr_read_restart,             &! read info from file needed to restart the time manager
    timemgr_restart                  ! restart the time manager

```

Public data for namelist input

```

character(len=32), public :: &
    calendar = 'GREGORIAN'      ! Calendar in date calculations ('NO_LEAP' or 'GREGORIAN')

integer, parameter :: uninit_int = -999999999 !This is private to this module

integer, public :: &
    dtime     = uninit_int,   &! timestep in seconds
    nestep    = uninit_int,   &! final timestep (or day if negative) number
    nelapse   = uninit_int,   &! number of timesteps (or days if negative) to extend a run
    start_ymd = uninit_int,   &! starting date for run in yearmmdd format
    start_tod = 0,           &! starting time of day for run in seconds
    stop_ymd  = uninit_int,   &! stopping date for run in yearmmdd format
    stop_tod  = 0,           &! stopping time of day for run in seconds
    ref_ymd   = uninit_int,   &! reference date for time coordinate in yearmmdd format
    ref_tod   = 0             ! reference time of day for time coordinate in seconds

```

1.18.1 timemgr_init (Source File: time_manager.F90)

Initialize the ESMF time manager.

NOTE - This assumes that the namelist variables have been set before this routine is called.

INTERFACE:

```
subroutine timemgr_init()
```

CONTENTS:

```
!-----
! Initialize error handling.
!-----
call esmf_errhandlersettype(esmf_err_return)
!-----
! Initialize calendar type.
!-----
cal = to_upper(calendar)
if ( trim(cal) == 'NO_LEAP' ) then
    cal_type = esmf_no_leap
else if ( trim(cal) == 'GREGORIAN' ) then
    cal_type = esmf_gregorian
else
    write(6,*),sub,: unrecognized calendar specified: ',calendar
    call endrun
end if
!-----
! Initialize timestep size.
!-----
if ( dtime == uninit_int ) then
    write(6,*),sub,: dtime must be specified in namelist'
    call endrun
end if
if ( mod(86400,dtime) /=0 ) then
    write(6,*),sub,: timestep must divide evenly into 1 day'
    call endrun
end if

step_size = esmf_timeinit(0, dtime, rc)
call chkrc(rc, sub//': error return from esmf_timeinit: setting step_size')
!-----
! Initialize start date.
!-----
if ( start_ymd == uninit_int ) then
    write(6,*),sub,: start_ymd must be specified in namelist'
    call endrun
end if
```

```

if ( start_tod == uninit_int ) then
    write(6,*)
    sub,: start_tod must be specified in namelist'
    call endrun
end if
start_date = esmf_dateinit(cal_type, start_ymd, start_tod, rc)
call chkrc(rc, sub//': error return from esmf_dateinit: setting start_date')
!-----
! Initialize reference date for time coordinate.
!-----
if ( ref_ymd /= uninit_int ) then
    ref_date = esmf_dateinit(cal_type, ref_ymd, ref_tod, rc)
else
    ref_date = esmf_dateinit(start_date, rc)
end if
call chkrc(rc, sub//': error return from esmf_dateinit: setting ref_date')
!-----
! Initialize stop date.
!-----
if ( stop_ymd /= uninit_int ) then
    stop_date = esmf_dateinit(cal_type, stop_ymd, stop_tod, rc)
else if ( nestep /= uninit_int ) then
    if ( nestep >= 0 ) then
        stop_date = esmf_dateincrementsec(start_date, dtime*nestep, rc)
    else
        stop_date = esmf_dateincrementday(start_date, -nestep, rc)
    end if
else if ( nelapse /= uninit_int ) then
    if ( nelapse >= 0 ) then
        stop_date = esmf_dateincrementsec(start_date, dtime*nelapse, rc)
    else
        stop_date = esmf_dateincrementday(start_date, -nelapse, rc)
    end if
else
    write(6,*)
    sub,: Must specify one of stop_ymd, nestep, or nelapse'
    call endrun
end if
call chkrc(rc, sub//': error return setting stop_date')
!-----
! Initialize a time manager.
!-----
tm_id = esmf_timemgrinit(step_size, start_date, stop_date, ref_date, rc)
call chkrc(rc, sub//': error return from esmf_timemgrinit')
!-----
! Calculation of ending timestep number (nestep) assumes a constant stepsize.
!-----
ntspday = 86400/dtime
diff = esmf_timeinit()
call esmf_datediff(start_date, stop_date, diff, islater, rc)

```

```

call chkrc(rc, sub//': error return from esmf_datediff calculating nestep')
call esmf_timeget(diff, ndays, nsecs, rc)
call chkrc(rc, sub//': error return from esmf_timeget calculating nestep')
nestep = ntspday*ndays + nsecs/dtime
if ( mod(nsecs,dtime) /= 0 ) nestep = nestep + 1
!-----
! Print configuration summary to log file (stdout).
!-----
if (masterproc) then
  call timemgr_print()
end if

```

1.18.2 timemgr_print (Source File: time_manager.F90)

Restart the ESMF time manager.

NOTE - Assumptions: 1) The namelist variables have been set before this routine is called. The stop date is the only thing that can be changed by the user on a restart.
2) Restart data have been read on the master process before this routine is called.
(timemgr_read_restart called from control/restart.F90::read_restart)

INTERFACE:

```
subroutine timemgr_restart()
```

1.18.3 timemgr_print (Source File: time_manager.F90)

Prints the time manager information

INTERFACE:

```
subroutine timemgr_print()
```

CONTENTS:

```

call esmf_tiemngrrestartwrite(tm_id, type, nstep, step_days, step_sec, &
                           start_ymd, start_tod, stop_ymd, stop_tod, ref_ymd, &
                           ref_tod, curr_ymd, curr_tod, rc)
call chkrc(rc, sub//': error return from esmf_tiemngrrestartwrite')

write(6,*)' ***** Time Manager Configuration *****'

if ( type == esmf_no_leap ) then
  cal = 'NO_LEAP'
else if ( type == esmf_gregorian ) then
  cal = 'GREGORIAN'

```

```

end if

write(6,*)' Calendar type:      ',trim(cal)
write(6,*)' Timestep size (seconds):  ', (step_days*86400 + step_sec)
write(6,*)' Start date (ymd tod):   ', start_ymd, start_tod
write(6,*)' Stop date (ymd tod):    ', stop_ymd, stop_tod
write(6,*)' Reference date (ymd tod):', ref_ymd, ref_tod
write(6,*)' Current step number:    ', nstep
write(6,*)' Ending step number:     ', nestep
write(6,*)' Current date (ymd tod):  ', curr_ymd, curr_tod

write(6,*)' ****'

```

1.18.4 advance_timestep (Source File: *time_manager.F90*)

Increment the timestep number.

INTERFACE:

```
subroutine advance_timestep()
```

CONTENTS:

```

call esmf_tiemgradvance(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiemgradvance')
!-----
! Set first step flag off.
!-----
first_restart_step = .false.

```

1.18.5 get_step_size (Source File: *time_manager.F90*)

Return the step size in seconds.

INTERFACE:

```
function get_step_size()
```

CONTENTS:

```

call esmf_tiemgrgetstepsize(tm_id, days, seconds, rc)
call chkrc(rc, sub//': error return from esmf_tiemgrgetstepsize')
get_step_size = 86400*days + seconds

```

1.18.6 get_nstep (Source File: *time_manager.F90*)

Return the timestep number.

INTERFACE:

```
function get_nstep()
```

CONTENTS:

```
get_nstep = esmf_timemgrgetnstep(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_timemgrgetnstep')
```

1.18.7 get_curr_day (Source File: *time_manager.F90*)

Return date components valid at end of current timestep with an optional offset (positive or negative) in seconds.

INTERFACE:

```
subroutine get_curr_date(yr, mon, day, tod, offset)
```

CONTENTS:

```
date = esmf_timemgrgetcurrdate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_timemgrgetcurrdate')

if (present(offset)) then
  if (offset > 0) then
    date = esmf_dateincrementsec(date, offset, rc)
    call chkrc(rc, sub//': error incrementing current date')
  else if (offset < 0) then
    off = esmf_timeinit(0, -offset, rc)
    call chkrc(rc, sub//': error setting offset time type')
    date = esmf_datedecrement(date, off, rc)
    call chkrc(rc, sub//': error decrementing current date')
  end if
end if

call esmf_dateget(date, ymd, tod, rc)
call chkrc(rc, sub//': error return from esmf_dateget')
yr = ymd/10000
mon = mod(ymd, 10000) / 100
day = mod(ymd, 100)
```

1.18.8 get_prev_date (Source File: *time_manager.F90*)

Return date components valid at beginning of current timestep.

INTERFACE:

```
subroutine get_prev_date(yr, mon, day, tod)
```

CONTENTS:

```
date = esmf_tiemmgrgetprevdate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiemmgrgetprevdate')

call esmf_dateget(date, ymd, tod, rc)
call chkrc(rc, sub//': error return from esmf_dateget')
yr = ymd/10000
mon = mod(ymd, 10000) / 100
day = mod(ymd, 100)
```

1.18.9 get_start_date (Source File: *time_manager.F90*)

Return date components valid at beginning of initial run.

INTERFACE:

```
subroutine get_start_date(yr, mon, day, tod)
```

CONTENTS:

```
date = esmf_tiemmgrgetstartdate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiemmgrgetstartdate')

call esmf_dateget(date, ymd, tod, rc)
call chkrc(rc, sub//': error return from esmf_dateget')
yr = ymd/10000
mon = mod(ymd, 10000) / 100
day = mod(ymd, 100)
```

1.18.10 get_ref_date (Source File: *time_manager.F90*)

Return date components of the reference date.

INTERFACE:

```
subroutine get_ref_date(yr, mon, day, tod)
```

CONTENTS:

```

date = esmf_tiememgrgetbasedate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrgetbasedate')

call esmf_dateget(date, ymd, tod, rc)
call chkrc(rc, sub//': error return from esmf_dateget')
yr = ymd/10000
mon = mod(ymd, 10000) / 100
day = mod(ymd, 100)

```

1.18.11 get_curr_time (Source File: *time_manager.F90*)

Return time components valid at end of current timestep. Current time is the time interval between the current date and the reference date.

INTERFACE:

```
subroutine get_curr_time(days, seconds)
```

CONTENTS:

```

cdate = esmf_tiememgrgetcurrdate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrgetcurrdate')

rdate = esmf_tiememgrgetbasedate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrgetbasedate')

call esmf_datediff(rdate, cdate, diff, islater, rc)
call chkrc(rc, sub//': error return from esmf_datediff')

call esmf_timeget(diff, days, seconds, rc)
call chkrc(rc, sub//': error return from esmf_timeget')
```

1.18.12 get_curr_calday (Source File: *time_manager.F90*)

Return calendar day at end of current timestep with optional offset. Calendar day 1.0 = 0Z on Jan 1.

INTERFACE:

```
function get_curr_calday(offset)
```

CONTENTS:

```

date = esmf_tiememgrgetcurrdate(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrgetcurrdate')
```

```

if (present(offset)) then
  if (offset > 0) then
    date = esmf_dateincrementsec(date, offset, rc)
    call chkrc(rc, sub//': error incrementing current date')
  else if (offset < 0) then
    off = esmf_timeinit(0, -offset, rc)
    call chkrc(rc, sub//': error setting offset time type')
    date = esmf_datedecrement(date, off, rc)
    call chkrc(rc, sub//': error decrementing current date')
  end if
end if

get_curr_calday = esmf_dategetfltdayofyear(date, rc)
call chkrc(rc, sub//': error return from esmf_dategetfltdayofyear')

```

1.18.13 is_end_curr_day (Source File: *time_manager.F90*)

Return true if current timestep is last timestep in current day.

INTERFACE:

```
function is_end_curr_day()
```

CONTENTS:

```

call get_curr_date(yr, mon, day, tod)
is_end_curr_day = (tod == 0)
```

1.18.14 is_end_curr_month (Source File: *time_manager.F90*)

Return true if current timestep is last timestep in current month.

INTERFACE:

```
function is_end_curr_month()
```

CONTENTS:

```

call get_curr_date(yr, mon, day, tod)
is_end_curr_month = (day == 1 .and. tod == 0)
```

1.18.15 is_first_step (Source File: *time_manager.F90*)

Return true on first step of initial run only.

INTERFACE:

```
function is_first_step()
```

CONTENTS:

```
nstep = esmf_tiememgrgetnstep(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrgetnstep')
is_first_step = (nstep == 0)
```

1.18.16 is_first_restart_step (Source File: *time_manager.F90*)

Return true on first step of restart run only.

INTERFACE:

```
function is_first_restart_step()
```

CONTENTS:

```
is_first_restart_step = first_restart_step
```

1.18.17 is_last_step (Source File: *time_manager.F90*)

Return true on last timestep.

INTERFACE:

```
function is_last_step()
```

CONTENTS:

```
is_last_step = esmf_tiememgrlaststep(tm_id, rc)
call chkrc(rc, sub//': error return from esmf_tiememgrlaststep')
```

1.18.18 timemgr_write_restart (Source File: *time_manager.F90*)

Write information needed on restart to a binary Fortran file. It is assumed that this routine is called only from the master proc if in SPMD mode.

INTERFACE:

```
subroutine timemgr_write_restart(ftn_unit)
```

CONTENTS:

```
call esmf_tiemngrrestartwrite(tm_id, rst_type, rst_nstep, rst_step_days, rst_step_sec, &
                             rst_start_ymd, rst_start_tod, rst_stop_ymd, rst_stop_tod, rst_ref_tod, &
                             rst_curr_ymd, rst_curr_tod, rc)
call chkrc(rc, sub//': error return from esmf_tiemngrrestartwrite')

write(ftn_unit, iostat=rc) rst_type, rst_nstep, rst_step_days, rst_step_sec, &
                           rst_start_ymd, rst_start_tod, rst_stop_ymd, rst_stop_tod, rst_ref_tod, &
                           rst_curr_ymd, rst_curr_tod

if (rc /= 0 ) then
  write (6,*) 'WRITE iostat= ',rc,' on i/o unit = ',ftn_unit
  call endrun
end if
```

1.18.19 timemgr_read_restart (Source File: *time_manager.F90*)

Read information needed on restart from a binary Fortran file. It is assumed that this routine is called only from the master proc if in SPMD mode.

INTERFACE:

```
subroutine timemgr_read_restart(ftn_unit)
```

CONTENTS:

```
read(ftn_unit, iostat=rc) rst_type, rst_nstep, rst_step_days, rst_step_sec, &
                           rst_start_ymd, rst_start_tod, rst_stop_ymd, rst_stop_tod, rst_ref_tod, &
                           rst_curr_ymd, rst_curr_tod

if (rc /= 0 ) then
  write (6,*) 'READ iostat= ',rc,' on i/o unit = ',ftn_unit
  call endrun
end if
```

1.18.20 chkrc (Source File: *time_manager.F90*)

Checks the return code for errors

INTERFACE:

```
subroutine chkrc(rc, mes)
```

CONTENTS:

```

if ( rc == esmf_success ) return
write(6,*) mes
call esmf_errprint(rc)
call endrun

```

1.19 Fortran: Module Interface time_module.F90 (Source File: time_module.F90)

Module contains variables that represent LDAS time

REVISION HISTORY:

```

15 Oct 1999: Paul Houser; Initial Version
26 Apr 2000: Brian Cosgrove; Fix for Alpha (Used NINT instead
              of DINT b/c of rounding/truncating issues)
01 May 2000: Brian Cosgrove; Correction in TIME2DATE subroutine,
              if hour=24, hour set to 0, day, month and year advanced

```

INTERFACE:

```
module time_module
```

ARGUMENTS:

```

type listime
    integer :: sss                      !starting second
    integer :: sdoy                     !starting day of year
    integer :: smn                      !starting minute
    integer :: shr                      !starting hour
    integer :: sda                     !starting day
    integer :: smo                      !starting month
    integer :: syr                      !starting year
    integer :: endcode                 !0=realtim, 1=specific date
    integer :: ess                      !ending second
    integer :: emn                      !ending minute
    integer :: edoy                     !ending day of year
    integer :: ehr                      !ending hour
    integer :: eda                      !ending day
    integer :: emo                      !ending month
    integer :: eyr                      !ending year
    integer :: ts                       !timestep (seconds)
    integer :: tscount                  !timestep count
    integer :: yyyyymmdd,hhmmss
    integer :: doy,yr,mo,da,hr,mn,ss !lis current model timing variables
    integer :: endtime                  !lis stop (0=continue time looping)
    integer :: pda                      !lis previous timestep day
    real*8 :: time                     !lis current model time in years
    real*8 :: etime                    !lis end time in years
    real :: gmt,egmt,sgmt
end type listime

```

Converts the date to year, month and day

INTERFACE:

```
subroutine date2days(time, yr, mo, da)
  implicit none
```

ARGUMENTS:

```
integer yr, mo, da
integer time
```

CONTENTS:

```
yr = time/10000
mo = (time-yr*10000)/100
da = (time-yr*10000-mo*100)
```

Converts time in seconds to hours, minutes and seconds

INTERFACE:

```
subroutine sec2time(sec, hr, mn, ss)
```

ARGUMENTS:

```
integer sec, hr, mn, ss
```

CONTENTS:

```
hr = sec/3600
mn = (sec-hr*3600)/60
ss = (sec-hr*3600-mn*60)
```

Updates the LDAS time variables according to the ESMF time

INTERFACE:

```
subroutine updatetime(lt)
  use lislog_module
  implicit none
```

ARGUMENTS:

```
type(listime)::lt
```

CONTENTS:

```

call date2time(lt%time,lt%doy,lt%gmt, &
               lt%yr,lt%mo,lt%da,lt%hr,lt%mn,lt%ss)
write(*,24)'GSFC-LIS time: ',lt%mo,'/',lt%da,'/', &
               lt%yr,lt%hr,':',lt%mn,':',lt%ss
write(79,24)'GSFC-LIS time: ',lt%mo,'/',lt%da,'/', &
               lt%yr,lt%hr,':',lt%mn,':',lt%ss

24 format(a16,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)

lt%yyyymmdd=(10000*lt%yr)+(100*lt%mo)+lt%da
lt%hhmmss=(10000*lt%hr)+(100*lt%mn)+lt%ss
!-----
! check for endtime
!-----
if(lt%endcode.eq.0)then !end at real-time date (tbd)
  write(*,*)'warning: do not know how to stop in real-time'
  write(79,*)'warning: do not know how to stop in real-time'
endif
if(lt%endcode.eq.1)then !end on date specified in lis.crd file
  call date2time(lt%etime,lt%edoy,lt%egmt, &
                 lt%eyr,lt%emo,lt%eda,lt%ehr,lt%emn,lt%ess)
  if(lt%time.ge.lt%etime)then
    lt%endtime=1
    write(*,*) 'GSFC-LDAS run completed'
    write(79,*) 'GSFC-LDAS run completed'
  endif
endif

return

```

advance (or retract) time variables a specified amount (a nonmodular version of ticktime.f.)

REVISION HISTORY:

```

1 oct 1999: jared entin; initial code
15 oct 1999: paul houser; significant f90 revision

```

INTERFACE:

```

subroutine tick(time,doy,gmt,yr,mo,da,hr,mn,ss,ts)
  implicit none

```

ARGUMENTS:

```

real*8 time
integer days(13)
integer yr,mo,da,hr,mn,ss,ts,doy
real gmt

```

CONTENTS:

```

143 format(a1,' yr',i6,' mo',i5,' dy',i5,' hr',i5, &
           ' mn',i6,' ss',i8,' ts',i8)
      ss=ss+ts
      do while(ss.gt.59)
          ss=ss-60
          mn=mn+1
      enddo
      do while(ss.lt.0)
          ss=ss+60
          mn=mn-1
      enddo
      do while(mn.gt.59)
          mn=mn-60
          hr=hr+1
      enddo

      do while(mn.lt.0)
          mn=mn+60
          hr=hr-1
      enddo
      do while(hr.gt.23)
          hr=hr-24
          da=da+1
      enddo

      do while(hr.lt.0)
          hr=hr+24
          da=da-1
      enddo

      if((mod(yr,4).eq.0.and.mod(yr,100).ne.0) &           !correct for leap year
         .or.(mod(yr,400).eq.0))then                  !correct for y2k
          days(2)=29
      else
          days(2)=28
      endif

      do while(da.gt.days(mo))
          da=da-days(mo)
          mo=mo+1
      enddo

      do while(da.lt.1)

          prvmo=mo-1
          if(mo.eq.1) prvmo=12

```

```

da=da+days(prvmo)

if(prvmo.eq.12) then
  mo=prvmo
  yr=yr-1
else
  mo=prvmo
endif
enddo
do while(mo.gt.12)
  mo=mo-12
  yr=yr+1
enddo

do while(mo.lt.1)
  mo=mo+12
  yr=yr-1
enddo
call date2time(time,doy,gmt,year,month,day,hour,minute,second)
return

```

determines time in years, based on year, month, day hour etc.. or reverse (date2time).

REVISION HISTORY:

```

15 oct 1999: paul houser; initial code
21 feb 2002: brian cosgrove; corrected leap year code line. days(2)
              was not being reset to 28 after leaving a leap year,
              it was staying 29

```

INTERFACE:

```

subroutine date2time(time,doy,gmt,year,month,day,hour,minute,second)

implicit none

```

ARGUMENTS:

```

integer year,month,day,hour,minute,second,yrdays,doy,days(13),k
real*8 time
real gmt

```

CONTENTS:

```

if((mod(year,4).eq.0.and.mod(year,100).ne.0) &      !correct for leap year
   .or.(mod(year,400).eq.0))then                  !correct for y2k
  yrdays=366
else

```

```

    yrdays=365
  endif

  doy=0
  do k=1,(mo-1)
    doy=doy+days(k)
  enddo
  doy=doy+da

  if(yrdays.eq.366.and.mo.gt.2)doy=doy+1

  time=(dfloat(yr)+((((((dfloat(ss)/60.d0)+dfloat(mn))/60.d0)+ &
    dfloat(hr))/24.d0)+dfloat(doy-1))/dfloat(yrdays))

  gmt=( ( float(ss)/60.0 ) +float(mn) ) /60.0)+float(hr)
  return

```

1.19.1 zterp.F90 (Source File: zterp.F90)

This subroutine is based, in part, on modified subroutines from Jean C. Morrill of the GSWP project. The program temporally interpolates time average or instantaneous data to that needed by the model at the current timestep. It does this by combining a linear interpolation approach with a solar zenith angle approach in a fashion suitable for use with data such as short wave radiation values. It cannot be used with input data points which are more than 24 hours apart. The program outputs two weights which can then be applied to the original data to arrive at the interpolated data. If IFLAG=0, then WEIGHT1 is the weight which should be applied to the time averaged data (from the time period which the model is currently in) to arrive at the interpolated value and weight 2 is not used at all. If IFLAG=1, then WEIGHT1 should be applied to the original instantaneous data located just prior to the model time step, and WEIGHT2 should be applied to the original instantaneous data located just after the model time step.

i.e. (IF IFLAG=0) interpolated data = (WEIGHT1 * time averaged data from time period that model is currently in)

i.e. (IF IFLAG=1) interp. data = (WEIGHT1*past data)+(WEIGHT2*future data)

REVISION HISTORY:

```

10/2/98 Brian Cosgrove
6/28/00 Brian Cosgrove; changed code so that it uses LDAS%UDEF and
not a hard-wired undefined value of -999.999. Also changed
call to ZTERP subroutine so that LDAS and GRID mod files
are brought in and can be accessed in this subroutine
2/27/01 Brian Cosgrove; Added czmodel into call for ZTERP subroutine
10/7/01 Urszula Jambor; Added conditional to prevent stop in code when
AGRMET data are available, but both endtime cos(zen.) are small.

```

INTERFACE:

```
subroutine zterp (iflag,lat,lon,btime,etime, &
                 mbtime,julianb,weight1,weight2,czbegdata, &
                 czenddata,czmodel,lis)
```

USES:

```
use lis_module      ! LDAS non-model-specific 1-D variables
implicit none
!INPUT PARAMETERS
type (lisdec) :: lis
integer         :: iflag      !Flag specifying if input data is
                           !time averaged or not
real            :: lat,lon   !Latitude and longitudes of current
                           !data point
real            :: mbtime    !Time of current time step Expects
                           !GMT time in hour fractions
integer         :: julianb   !Julian day upon which BTIME falls
real            :: interval
real            :: btime     !beginning time of orig. avg. data (IFLAG=0) or
                           !time of original instantaneous data point which
                           !is located at or just prior to the current model
                           !time step (IFLAG=1). Expects GMT time in hour
                           !fractions. i.e., 6:30 Z would be 6.5.
real            :: etime      !Ending time of orig. avg. data (IFLAG=0) or
                           !time of original instantaneous data point which
                           !is located at or just after the current model
                           !time step (IFLAG=1). Expects GMT time in hour
                           !fractions. i.e., 6:30 Z would be 6.5.
real            :: weight1    !weight applied to original time averaged
                           !data (IFLAG=0) or
                           !weight applied to orig instantaneous data point
                           !located just prior to the current model time step
real            :: weight2    !weight applied to orig instantaneous
                           !data point located just after the current model
                           !time step (IFLAG=1)
                           !If IFLAG=0, then this weight is meaningless and
                           !should not be used
```

CONTENTS:

```
!-----
! This section contains hardwired data that will be supplied by main program.
! These values were chosen arbitrarily and exist simply to check the
! functioning of the program.
!
! Initialize variables
!-----
i=1
totangle=0
```

```

weighte=lis%d%udef
weightb=lis%d%udef
weight1=lis%d%udef
weight2=lis%d%udef
czbegdata=lis%d%udef
czenddata=lis%d%udef
czmodel=lis%d%udef
gmt=btime
juliane=julianb
julianmb=julianb
juliantemp=julianb

if (mbtime.lt.btime) julianmb=julianmb+1
if (etime.le.btime) juliane=juliane+1
!-----
! First case, IFLAG=0 (Time average input, instantaneous output)
! Compute time interval, here arbitrarily divided into 36 parts
!-----
if (iflag.eq.0) then
  call localtime (mbtime,lon,lhour,zone)
  call coszenith(lon,lat,lhour,zone,julianmb,czmodel)
  if (czmodel.eq.0) then
    weight1=0
    goto 818
  endif

  if (etime.gt.btime) then
    interval = ((etime-btime)/36.0)
  elseif (etime.lt.btime) then
    interval = (((24-btime)+(etime))/36.0)
  else
    interval=24.0/36.0
  endif
!-----
!      Compute cosine of zenith angle for each time interval
!-----
do while (i.le.37)
  if ((gmt+interval).lt.24) then
    gmt=gmt+interval
  else
    gmt=(interval-(24-gmt))
    juliantemp=juliantemp+1
  endif
  call localtime (gmt,lon,lhour,zone)
  call coszenith(lon,lat,lhour,zone, &
                juliantemp,czavgdata)
  totangle=totangle+czavgdata
  i=i+1

```

```

    enddo
!-----
!      Compute average cosine of zenith angle and also
!      weight which will be applied to original data (WEIGHT1
!-----
    avgangle=(totangle/37.0)
    if (avgangle.eq.0) then
        weight1=0
    else
        weight1=(czmodel/avgangle)
    endif
    endif
!-----
!      Second case: IFLAG=1 (instantaneous input and output)
!-----
    if (iflag.eq.1) then
!-----
!      Compute local times and cosine (zenith angle)
!-----
        call localtime (btime,lon,lctime,zonebtime)
        if (lctime.gt.btime) julianb=julianb-1
        call localtime (etime,lon,letime,zoneetime)
        if (letime.gt.etime) juliane=juliane-1
        call localtime (mbtime,lon,lmbtime,zonembtime)
        if (lmbtime.gt.mbtme) julianmb=julianmb-1
        call coszenith (lon,lat,lctime,zonebtime, &
                        julianb,czbegdata)
        call coszenith (lon,lat,letime,zoneetime, &
                        juliane,czenddata)
        call coszenith (lon,lat,lmbtime,zonembtime, &
                        julianmb,czmodel)
!-----
!      Decision tree to deal with contingencies
!      If COS(zenith angle at current model time =0, weight =0
!      If COS(zenith angle =0 at beg. and end times, PROBLEM, STOP
!      Otherwise use beginning and ending data to calculate weight
!-----
    if (czmodel.le.0.01) then
        weight1=0
        weight2=0
    else
        if ((czbegdata.gt.0.01).or.(czenddata.gt.0.01)) then
            if (czbegdata.le.0.01) then
                weight1=0
                weight2=(czmodel/czenddata)
            endif
            if (czenddata.le.0.01) then
                weight1=(czmodel/czbegdata)
            endif
        endif
    endif

```

```

        weight2=0
    endif
    if((czenddata.gt.0.01).and. &
       (czbegdata.gt.0.01))then

        if (btime.le.mbtme) then
            diffbm=mbtme-btime
        else
            diffbm=24-btime+mbtme
        endif

        if (etime.ge.mbtme) then
            diffem=etime-mbtme
        else
            diffem=24-mbtme+etime
        endif

        if (etime.gt.btime) then
            diffeb=etime-btime
        elseif (etime.eq.btime) then

            diffeb=24
        else
            diffeb=24-btime+etime
        endif
        weighte=(diffbm/diffeb)
        weightb=(diffem/diffeb)

        weight1=((czmodel/czbegdata)*weightb)
        weight2=((czmodel/czenddata)*weighte)

    endif
    else
        write(79,*)"no data to interpolate to/from"
        write(79,*)"beginning and ending data both = 0"
        write(79,*)"stopping!!!"

        print*, 'no data to interpolate to/from'
        print*, 'beginning and ending data both = 0'
        print*, 'stopping!!!'

        call endrun
    endif
endif
endif
818 continue
return

```

1.19.2 coszenith (Source File: zterp.F90)

- 1) Day angle (GAMMA)
- 2) Solar DEClination
- 3) Equation of time
- 4) Local apparent time
- 5) Hour angle
- 6) Cosine of zenith angle

All equations come from "An Introduction to Solar Radition" By Muhammad Iqbal, 1983.

INTERFACE:

```
subroutine coszenith (lon,latd,lhour,zone,julian,czenith)
```

```
    implicit none
```

ARGUMENTS:

integer :: zone	! time zone (1-24) gmt=12
integer :: julian	! julian day
real :: czenith	! cosine of zenith angle (radians)
real :: dec	! solar declination (radians)
real :: et	! equation of time (minutes)
real :: gamma	! day angle (radians)
real :: latime	! local apparent time
real :: lcorr	! longitudinal correction
real :: lhour	! local standard time
real :: lon	! local longitude (deg)
real :: llat	! local latitude in radians
real :: latd	! local latitude in degrees
real :: ls	! standard longitude (deg)
real :: omegad	! omega in degrees
real :: omega	! omega in radians
real :: pi	! universal constant pi [-]
real :: zenith	! zenith angle(radians)

CONTENTS:

```
!-----
! Neither ZENITH nor ZEND are necessary for this program.
! I originally used them as checks, and left them here in
! case anyone else had a use for them.
!
!     1) Day angle GAMMA (radians) page 3
!-----
```

```
pi= 3.141592           ! universal constant pi
gamma=2*pi*(julian-1)/365.
!-----
```

```

! 2) Solar declination (assumed constant for a 24 hour period) page 7
! in radians
!-----
dec=(0.006918-0.399912*cos(gamma)+0.070257*sin(gamma) &
      -0.006758*cos(2*gamma)+0.000907*sin(2*gamma) &
      -0.002697*cos(3*gamma)+0.00148*sin(3*gamma))

!-----
! maximum error 0.0006 rad (<3'), leads to error of less than 1/2 degree
! in ZENITH angle
! 3) Equation of time page 11
!-----

et=(0.000075+0.001868*cos(gamma)-0.032077*sin(gamma) &
      -0.014615*cos(2*gamma)-0.04089*sin(2*gamma))*229.18

!-----
! 4) Local apparent time page 13
!
! LS      standard longitude (nearest 15 degree meridian)
! LON     local longitude
! LHOUR   local standard time
! LATIME  local apparent time
! LCORR   longitudunal correction (minutes)
!-----
ls=((zone-1)*15)-180.
lcorr=4.*(ls-lon)*(-1)
latime=lhour+lcorr/60.+et/60.
if (latime.lt.0.) latime=latime+24
if (latime.gt.24.) latime=latime-24

!-----
! 5) Hour angle OMEGA page 15
!
! hour angle is zero at noon, postive in the morning
! It ranges from 180 to -180
!
! OMEGAD is OMEGA in degrees, OMEGA is in radians
!-----
OMEGAD=(LATime-12.)*(-15.)
OMEGA=OMEGAD*PI/180.

!-----
! 6) Zenith angle page 15
!
! CZENITH cosine of zenith angle (radians)
! LATD=local latitude in degrees
! LLAT=local latitude in radians
!-----
llat=latd*pi/180.
czenith=sin(dec)*sin(llat)+cos(dec)*cos(llat)*cos(omega)
czenith=amax1(0.,czenith)

```

```
zenith=asin(czenith)
return
```

1.19.3 localtime (Source File: zterp.F90)

Calculates the local time based on GMT

INTERFACE:

```
subroutine localtime (gmt,lon,lhour,zone)
```

ARGUMENTS:

real:: gmt	! GMT time (0-23)
real:: lon	! longitude in degrees
real:: change	! the change in number of hours between
real:: lhour	! local hour (0-23) 0= midnight, 23= 11:00 p.m.
integer:: i	! working integer
integer:: zone	! time zone (1-24)

CONTENTS:

```
!-----
! Determine into which time ZONE (15 degree interval) the
! longitude falls.
!-----
do i=1,25
  if (lon.lt.(-187.5+(15*i))) then
    zone=i
    if (zone.eq.25) zone=1
    go to 60
  end if
end do
!-----
! Calculate change (in number of hours) from GMT time to
! local hour. Change will be negative for zones < 13 and
! positive for zones > 13.
! There is also a correction for L HOUR < 0 and L HOUR > 23
! to L HOUR between 0 and 23.
!-----
60 if (zone.lt.13) then
  change=zone-13
  lhour=gmt+change
elseif (zone.eq.13) then
  lhour=gmt
else
  change=zone-13
```

```

lhour=gmt+change
end if
if (lhour.lt.0) lhour=lhour+24
if (lhour.gt.23) lhour=lhour-24
return

```

1.20 Fortran: Module Interface geosdomain_module.F90 (Source File: geosdomain_module.F90)

Contains routines and variables that define the native domain for GEOS model forcing.

INTERFACE:

```
module geosdomain_module
```

USES:

```
use geosdrv_module
```

ARGUMENTS:

```
type(geosdrvdec) :: geosdrv
integer :: mi
```

1.20.1 defnatgeos.F90 (Source File: geosdomain_module.F90)

Defines the kgds array describing the native forcing resolution for GEOS data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnatgeos(kgdsi)
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

CONTENTS:

```
call readgeoscrd(geosdrv,kgdsi)
mi = geosdrv%ncold*geosdrv%nrold
```

1.21 Fortran: Module Interface geosdrv_module.F90 (Source File: geosdrv_module.F90)

Module containing runtime specific GEOS variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module geosdrv_module
```

ARGUMENTS:

```
type geosdrvdec
    integer :: ncold, nrold      !AWIPS 212 dimensions
    integer :: nmif
    character*40 :: geosdir     !GEOS Forcing Directory
    real*8 :: geostime1,geostime2
end type geosdrvdec
```

1.22 Fortran: Module Interface geosopendap_module.F90 (Source File: geosopendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O specific to GEOS forcing routines

REVISION HISTORY:

```
10 Jul 2003; James Geiger Initial Specification
22 Dec 2003; Sujay Kumar Separated geos specific routines from the main
               module
```

INTERFACE:

```
module geosopendap_module
#ifndef ( defined OPENDAP )
```

USES:

```
use geosdomain_module, only : geosdrv
use ESMF_TimeMgmtMod
use lisdrv_module, only : lis, grid, gindex
use grid_spmdMod,   only : gdi, gdisp
use tile_spmdMod,   only : di_array
use spmdMod

implicit none
type(esmf_date), save :: geos_ref_date !Reference date for GEOS forcing data
```

```

integer :: geos_slat      !Southern latitude boundary for each subdomain (for
integer :: geos_nlat      !Northern latitude boundary for each subdomain (for
integer :: geos_nc       !Number of columns for each subdomain (for GEOS)
integer :: geos_nr       !Number of rows for each subdomain (for GEOS)
integer :: input_slat    !Southern latitude boundary for the native domain
integer :: input_nlat    !Northern latitude boundary for the native domain
character*4 :: cgeos_slat !character representation of geos$-$slat
character*4 :: cgeos_nlat !character representation of geos$-$nlat
character*3 :: ciam       !character representation of processor id
integer :: grid_offset   !Global to local grid mapping offset
integer :: fnroffset

#if ( defined ABSOFT )
    character*15 :: opendap_geos_home = '/home/jim/'
#else
    character*2 :: opendap_geos_home = './'
#endif
contains
%%%%%%%%%%%%%%%
\mbox{}\\rulefill\\

\subsubsection{opendap\_geos\_init (Source File: geosopendap\_module.F90)}
```

Initializes the GEOS-GDS variables

\bigskip{\sf INTERFACE:
\begin{verbatim} subroutine opendap_geos_init()

CONTENTS:

```
call init_geos_vars()
call reset_geos_filepaths()
```

1.22.1 reset_geos_filepaths (Source File: geosopendap_module.F90)

Resets input data filenames for GEOS forcing for execution through GDS

INTERFACE:

```
subroutine reset_geos_filepaths()
```

CONTENTS:

```
geosdrv%geosdir      = trim(opendap_geos_home)//trim(adjustl(ciam))///'//geosdrv%geosd
```

1.22.2 init_geos_vars (Source File: geosopendap_module.F90)

Computes domain decomposition for native as well as interpolated domains for input GEOS forcing

INTERFACE:

```
subroutine init_geos_vars()
```

CONTENTS:

```
select case (lis%d%domain)
case(1)
    print*, 'Error!  Cannot handle ndas.'
    stop 999
case(2)
    res = 250
    center = 875
    bottom = -59875
case(3)
    print*, 'Error!  Cannot handle 2x2.5.'
    stop 999
case(4)
    res = 1000
    center = 500
    bottom = -59500
case(5)
    res = 500
    center = 750
    bottom = -59750
case(6)
    res = 50
    center = 975
    bottom = -59975
case DEFAULT
    print*, "Select domain size (1,2,3,4,5,6)"
    stop 999
end select

call set_geos_lat(geos_slat,geos_nlat,input_slat,input_nlat)

lis%d%ngrid = gdi(iam)
lis%d%nch   = di_array(iam)

geos_nc      = 360
geos_nr      = ( geos_nlat - geos_slat + 1 )

fnroffset = geos_slat - 1
grid_offset = tile(1)%index-1
```

```

write(ciam, '(i3)') iam
write(cgeos_slat, '(i4')') geos_slat
write(cgeos_nlat, '(i4')') geos_nlat

print*,'DBG: geos_init -- geos_slat', geos_slat, '(, iam, ')
print*,'DBG: geos_init -- geos_nlat', geos_nlat, '(, iam, ')
print*,'DBG: geos_init -- ngrid', lis%d%ngrid, '(, iam, ')
print*,'DBG: geos_init -- glbngrid', lis%d%glbngrid, '(, iam, ')
print*,'DBG: geos_init -- ncold', geosdrv%ncold, '(, iam, ')
print*,'DBG: geos_init -- nrold', geosdrv%nrold, '(, iam, ')
print*,'DBG: geos_init -- lnc', lis%d%lnc, '(, iam, ')
print*,'DBG: geos_init -- lnr', lis%d%lnr, '(, iam, ')
print*,'DBG: geos_init -- fnroffset', fnroffset, '(, iam, ')
print*,'DBG: geos_init -- grid_offset', grid_offset, '(, iam, ')
print*,'DBG: geos_init -- cgeos_slat ', cgeos_slat, '(, iam, ')
print*,'DBG: geos_init -- cgeos_nlat ', cgeos_nlat, '(, iam, ')
print*,'DBG: geos_init -- ciam ', ciam, '(, iam, ')
print*,'DBG: geos_init -- geos_nc', geos_nc, '(, iam, ')
print*,'DBG: geos_init -- geos_nr', geos_nr, '(, iam, ')
line = '"/ciam//"
print*,'DBG: geos_init -- line', line, '(, iam, ')

```

1.22.3 def_kgds (Source File: geosopendap_module.F90)

Initializes the kgds array for GEOS-GDS runs.

INTERFACE:

```
subroutine def_kgds(kgdsi)
```

CONTENTS:

```

kgdsi(1) = 0
kgdsi(2) = geos_nc
kgdsi(3) = geos_nr
kgdsi(4) = input_slat
kgdsi(7) = input_nlat
kgdsi(5) = -180000
kgdsi(6) = 128
kgdsi(8) = 179000
kgdsi(9) = 1000
kgdsi(10) = 1000
kgdsi(20) = 255

```

1.22.4 init_geos_ref_date (Source File: geosopendap_module.F90)

Initializes the reference date for GEOS forcing data

INTERFACE:

```
subroutine init_geos_ref_date()
```

CONTENTS:

```
geos_ref_date = esmf_dateinit(esmf_no_leap, ref_ymd, ref_tod, rc)
```

1.22.5 get_geos_index (Source File: geosopendap_module.F90)

Computes the time-based index for GEOS forcing data

INTERFACE:

```
function get_geos_index(offset)
  implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: offset ! offset from current date in hours
```

CONTENTS:

```
if ( ref_data_uninit ) then
  print*, 'DBG: get_geos_index -- initializing ref date', iam
  call init_geos_ref_date()
  ref_data_uninit = .false.
endif
ymd = ( lis%t%yr * 10000 ) + ( lis%t%mo * 100 ) + lis%t%da
tod = ( lis%t%hr * 3600 ) + ( lis%t%mn * 60 ) + lis%t%ss
current_date = esmf_dateinit(esmf_no_leap, ymd, tod, rc)

diff = esmf_timeinit()
call esmf_datediff(current_date, geos_ref_date, diff, islater, rc)
call esmf_timeget(diff, ndays, nsecs, rc)

get_geos_index = ( (ndays * 24) + (nsecs / 3600) + offset ) / 3 + 1
print*, 'DBG: get_geos_index -- get_geos_index = ', get_geos_index, iam
```

1.22.6 set_geos_lat (Source File: geosopendap_module.F90)

Computes the latitudes of the decomposed domain for GEOS forcing data

INTERFACE:

```
subroutine set_geos_lat(slat, nlat, islat, inlat)
    implicit none
```

OUTPUT PARAMETERS:

```
    integer, intent(out) :: slat, nlat, islat, inlat
```

CONTENTS:

```
!-----
! Set southern latitude index
!-----
print*, 'DBG: set_geos_lat -- grid(1)%lat', grid(1)%lat, ', iam, ')
lat = grid(1)%lat
if ( lat < 0.0 ) then
    slat = int(lat) - 1 ! E.g. map -54.5 \to -55
else
    slat = int(lat)      ! E.g. map 40.5 \to 40
endif
if ( slat < -90 ) then
    print*, 'ERR: set_geos_lat -- Setting slat = -90', &
        ', iam, ')
    slat = -90
endif
!-----
! Set southern latitude boundary
!-----
islat = 1000 * slat
!-----
! Set northern latitude index
!-----
print*, 'DBG: set_geos_lat -- grid(gdi(iam))', &
    gdi(iam), grid(gdi(iam))%lat, ', iam, ')
lat = grid(gdi(iam))%lat
if ( lat < 0.0 ) then
    nlat = int(lat)      ! E.g. map -10.5 \to -10
else
    nlat = int(lat) + 1 ! E.g. map 10.5 \to 11
endif
if ( nlat > 90 ) then
    print*, 'ERR: set_geos_lat -- Setting nlat = 90', &
        ', iam, ')
    nlat = 90
endif
```

```
!-----
! Set northern latitude boundary
!-----
inlat = 1000 * nlat

slat = slat + 90 + 1 ! map [-90,90] \to [1,181]
nlat = nlat + 90 + 1 ! map [-90,90] \to [1,181]
slat = 1
nlat = 181
islat = -90000
inlat = 90000
```

1.22.7 getgeos.F90 (Source File: getgeos.F90)

Opens, reads, and interpolates GEOS forcing.

TIME1 = most recent past data

TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.23 Core Functions of getgeos

tick Determines GEOS data times

geosfile Puts together appropriate file name for 3 hour intervals

readgeos Interpolates GEOS data to LDAS grid

REVISION HISTORY:

- 1 Oct 1999: Jared Entin; Initial code
- 25 Oct 1999: Jared Entin; Significant F90 Revision
- 11 Apr 2000: Brian Cosgrove; Fixed name construction error
in Subroutine ETA6HFILE
- 27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
data with opposite sign convention from recent shortwave data.
Added capability to use time averaged shortwave & longwave data
Altered times which are passed into ZTERP--used to be GMT1
and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
- 30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
- 17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
- 13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
- 5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.

INTERFACE:

```
subroutine getgeos()
```

USES:

```
use lisdrv_module, only : lis
use time_manager
use spmdMod
use tile_spmdMod
use time_module, only : tick,date2time
use baseforcing_module, only: glbdata1,glbdata2
use geosdomain_module, only : geosdrv
```

CONTENTS:

```
if ( masterproc ) then
    nstep = get_nstep()
endif
#ifndef ( defined OPENDAP )
    call MPI_BCAST(nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
#endif
!-----
! Determine the correct number of forcing variables
!-----
if ( nstep .eq. 0) then
    nforce = geosdrv%nmif
else
    nforce = lis%f%nf
endif
lis%f%findtime1=0
lis%f%findtime2=0
movetime=0
!-----
! Determine Required GEOS Data Times
! (The previous hour & the future hour)
!-----
yr1=lis%t%yr      !Time now
mo1=lis%t%mo
da1=lis%t%da
hr1=lis%t%hr
mn1=lis%t%mn
ss1=0
ts1=0

call tick(timenow,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

griduptime = 0.0
if (lis%f%gridchange==1) then
    yr1 = 2002 !grid update time
    mo1 = 11
```

```

da1 = 01
hr1 = 0; mn1 = 0; ss1 = 0
call date2time(griduptime,updoy,upgmt,yr1,mo1,da1,hr1,mn1,ss1)
endif

yr1=lis%t%yr      !Previous Hour
mo1=lis%t%mo
da1=lis%t%da
hr1=3*((lis%t%hr)/3)
mn1=0
ss1=0
ts1=0
call tick(time1,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

yr2=lis%t%yr      !Next Hour
mo2=lis%t%mo
da2=lis%t%da
hr2=3*((lis%t%hr)/3)
mn2=0
ss2=0
ts2=3*60*60

call tick(time2,doy2,gmt2,yr2,mo2,da2,hr2,mn2,ss2,ts2)
if(timenow.gt.geosdrv%geostime2) then
    movetime=1
    lis%f%findtime2=1
endif

if ( nstep.eq.0 .or. nstep.eq.1 .or.lis%f%rstflag.eq.1 ) then
    lis%f%findtime1=1
    lis%f%findtime2=1
    glbdata1 = 0
    glbdata2 = 0
    movetime=0
    lis%f%rstflag = 0
endif
lis%f%shortflag=2          !Time averaged SW
lis%f%longflag=2           !Time averaged LW
!-----
! Establish geostime1
!-----
if (lis%f%findtime1==1) then
    order=1
    ferror = 0
    try = 0
    ts1 = -24*60*60
    do
        if ( ferror /= 0 ) then

```

```

        exit
    end if
    try = try+1
    if(lis%f%gridchange.eq.1) then
        if(time1>griduptime) then
            geosdrv%ncold = 288
        endif
    endif
    call geosfile(name,geosdrv%geosdir,yr1,mo1,da1,hr1,geosdrv%ncold)
    call readgeos(order,name,lis%d,lis%t,ferror)
    if ( ferror == 1 ) then
!---
! successfully retrieved forcing data
!---
        geosdrv%geostime1=time1
    else
!---
! ferror still=0, so roll back one day
!---
        call tick(dumbtime1,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)
    end if
    if ( try > ndays ) then
        print *, 'ERROR: GEOS data gap exceeds 10 days on file 1'
        call endrun
    end if
    end do
endif
if(movetime.eq.1) then
    geosdrv%geostime1=geosdrv%geostime2
    lis%f%findtime2=1
    do f=1,nforce
        do c=1,lis%d%ngrid
            glbdata1(f,c)=glbdata2(f,c)
        enddo
    enddo
endif
if(lis%f%findtime2.eq.1) then
    order=2
    ferror = 0
    try = 0
    ts2 = -24*60*60
    do
        if ( ferror /= 0 ) exit
        try = try+1
        if(lis%f%gridchange==1) then
            if(time2>=griduptime) then
                geosdrv%ncold = 288
            endif
        endif
    end do
endif

```

```

        endif
        call geosfile(name,geosdrv%geosdir,yr2,mo2,da2,hr2,geosdrv%ncold)
        call readgeos(order,name,lis%d,lis%t,ferror)

        if ( ferror == 1 ) then
!-----
! successfully retrieved forcing data
!-----
        geosdrv%geostime2=time2
        else
!-----
! ferror still=0, so roll back one day
!-----
        call tick(dumbtime2,doy2,gmt2,yr2,mo2,da2,hr2,mn2,ss2,ts2)
        end if
        if ( try > ndays ) then
            print *, 'ERROR: GEOS data gap exceeds 10 days on file 2'
            call endrun
        end if
        end do
    endif

84 format('now',i4,4i3,2x,'pvt ',a22,' nxt ',a22)
    if ((lis%f%gridchange==1).and.(geosdrv%ncold==288)) then
        elevfile = lis%p%elevfile
        fpart1 = elevfile(1:21)
        fpart2 = elevfile(23:40)
        lis%p%elevfile = trim(fpart1) // "4" // trim(fpart2)
        print*, 'Use newer elevation difference file: ', lis%p%elevfile
        write(79,*) 'Transitioned from GEOS3 to GEOS4 grid dimensions.'
        lis%f%gridchange=0
    endif
    return

```

1.23.1 geosfile (Source File: getgeos.F90)

This subroutine puts together GEOS file name

INTERFACE:

```

subroutine geosfile(name,geosdir,yr,mo,da,hr, ncold)

implicit none

```

INPUT PARAMETERS:

```
character*40 geosdir
integer yr,mo,da,hr,ncold
```

OUTPUT PARAMETERS:

```
character*80 name
```

CONTENTS:

```
91 format(a4,i3,a11,i3)
92 format(80a1)
93 format(a80)
94 format(i4,i2,i2,a2)
95 format(10a1)
96 format(a40)
97 format(a8)
98 format(a1,i4,i2,a1)
99 format(8a1)
!-----
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----
uyr=yr
umo=mo
uda=da
uhr = 3*(hr/3) !hour needs to be a multiple of 3 hours
!-----
! Determine initcode for the hour of the forecast file
! If the time is 12 or later the file is time stamped
! with the next day. So check for that first
!-----
if(uhr<3)then
  initcode = '00'
elseif(uhr<6)then
  initcode = '03'
elseif(uhr<9)then
  initcode = '06'
elseif(uhr<12)then
  initcode = '09'
elseif(uhr<15)then
  initcode = '12'
elseif(uhr<18)then
  initcode = '15'
elseif(uhr<21)then
  initcode = '18'
elseif(uhr<24)then
  initcode = '21'
endif
```

```

write(UNIT=temp,FMT='(A40)') geosdir
read(UNIT=temp,FMT='(80A1)') (fbase(i),i=1,80)

write(UNIT=temp,FMT='(a1,i4,i2,a1)') '/ ,uyr,umo,'/
read(UNIT=temp,FMT='(8A1)') fdir
do i=1,8
  if(fdir(i).eq.(' ')) fdir(i)='0'
enddo

write(UNIT=temp,FMT='(i4,i2,i2,a2)') uyr,umo,uda,initcode
read(UNIT=temp,FMT='(10A1)') ftime
do i=1,10
  if(ftime(i).eq.(' ')) ftime(i)='0'
enddo

if(ncold==360) then
  write(UNIT=temp,FMT='(A8)') '.GEOS323'
  read(UNIT=temp,FMT='(80A1)') (fsubs(i),i=1,8)
else
  write(UNIT=temp,FMT='(A6)') '.GEOS4'
  read(UNIT=temp,FMT='(80A1)') (fsubs(i),i=1,6)
endif
c=0
do i=1,80
  if(fbase(i).eq.(' ').and.c.eq.0) c=i-1
enddo

if (ncold==360) then
  write(UNIT=temp,FMT='(80a1)') (fbase(i),i=1,c),(fdir(i),i=1,8), &
  (ftime(i),i=1,10),(fsubs(i),i=1,8)
else
  write(UNIT=temp,FMT='(80a1)') (fbase(i),i=1,c),(fdir(i),i=1,8), &
  (ftime(i),i=1,10),(fsubs(i),i=1,6)
endif

read(UNIT=temp, FMT='(a80)') name
write(679,*)hr,name
return

```

1.23.2 readgeoscrd.F90 (Source File: readgeoscrd.F90)

Routine to read GEOS specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readgeoscrd(geosdrv,kgdsi)
```

USES:

```
use geosdrv_module
#if ( defined OPENDAP )
    use geosopendap_module, only : opendap_geos_init, &
        def_kgds
#endif
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=geos)
print*, 'Using GEOS forcing'
print*, 'GEOS forcing directory : ', geosdrv%GEOSDIR
geosdrv%geostime1 = 3000.0
geosdrv%geostime2 = 0.0

#if ( defined OPENDAP )
    call opendap_geos_init()
    call def_kgds(kgdsi)
#endif
#ifndef (! defined OPENDAP)
    kgdsi(1) = 0
    kgdsi(2) = geosdrv%ncold
    kgdsi(3) = geosdrv%nrold
    kgdsi(4) = -90000
    kgdsi(7) = 90000
    kgdsi(5) = -180000
    kgdsi(6) = 128
    kgdsi(8) = 179000
    kgdsi(9) = 1000
    kgdsi(10) = 1000
    kgdsi(20) = 255
#endif
close(11)
```

1.23.3 *readgeos.F90* (Source File: *readgeos.F90*)

Reads in GEOS data and performs interpolation to the LDAS domain.

1.24 Core Functions of *readgeos*

polates0 Interpolates GEOS data to LDAS grid using polates0 (bilinear)

GEOS FORCING VARIABLES (unless noted, fields are 3-hr upstream averaged):

1. T 2m Temperature interpolated to 2 metres [K]
2. q 2m Instantaneous specific humidity interpolated to 2 metres[kg/kg]
3. radswg Downward shortwave flux at the ground [W/m²]
4. lwgdown Downward longwave radiation at the ground [W/m²]
5. u 10m Instantaneous zonal wind interpolated to 10 metres [m/s]
6. v 10m Instantaneous meridional wind interpolated to 10 metres[m/s]
7. ps Instantaneous Surface Pressure [Pa]
8. preacc Total precipitation [mm/s]
9. precon Convective precipitation [mm/s]
10. albedo Surface albedo (0-1)

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Added read statements for forcing interpolation
17 Apr 2001: Jon Gottschalck; Added code to perform initialization of
               Mosaic with GEOS forcing and new intp. scheme
14 Aug 2001: Urszula Jambor; Added ferror flag as a routine argument
07 Dec 2001: Urszula Jambor; Began used of LDAS$$LDAS_KGDS array

```

INTERFACE:

```
subroutine readgeos(order,name,ld,tscount,ferror)
```

USES:

```

use lis_module          ! LDAS non-model-specific 1-D variables
use spmdMod
use baseforcing_module, only: glbdata1, glbdata2
use def_ipMod, only : w110,w120,w210,w220,n110,n120,n210,n220, &
    rlat0,rlon0
use geosdomain_module, only : geosdrv,mi
    w113,w123,w213,w223,n113,n123,n213,n223,rlat3,rlon3
#if ( defined OPENDAP )
    use tile_spmdMod
    use geosopendap_module, only : geos_slat, geos_nlat, &
        cgeos_slat, cgeos_nlat,geos_nr,geos_nc,&
        get_geos_index
    use opendap_module, only : ciam, parm_nc, parm_nr, &
        nroffset
#endif
    use lisdrv_module, only : gindex ! LDAS non-model-specific 1-D variables

```

CONTENTS:

```
#if ( defined OPENDAP )
    integer :: nr_index, nc_index
```

```

integer :: nr_dom, nc_dom
integer :: geos_index
character*4 :: cgeos_index
real, allocatable :: debuggeos0(:,:)
real, allocatable :: debuggeos1(:,:)
integer :: ngrid1
integer :: nmif1
integer :: ierr, status(MPI_STATUS_SIZE)
nr_index = geos_nr
nc_index = geos_nc
nr_dom = parm_nr!ld%lnr
nc_dom = parm_nc!ld%lnc
#else
integer :: nr_index, nc_index
integer :: nr_dom, nc_dom
integer :: fnroffset=0
integer :: nroffset=0
integer :: geos_nc=0
nr_index = geosdrv%nrold
nc_index = geosdrv%ncold
nr_dom   = ld%lnr
nc_dom   = ld%lnc
#endif

allocate(tempvar(nc_index,nr_index,geosdrv%nmif), stat=ios)
call check_error(ios,'Error allocating tempvar.',iam)

allocate(tempgeos(ld%ngrid,geosdrv%nmif), stat=ios)
call check_error(ios,'Error allocating tempgeos.',iam)

allocate(f(nc_index*nr_index), stat=ios)
call check_error(ios,'Error allocating f.',iam)

allocate(go(nc_dom*nr_dom), stat=ios)
call check_error(ios,'Error allocating go.',iam)

allocate(gmask(nc_index*nr_index), stat=ios)
call check_error(ios,'Error allocating gmask.',iam)

allocate(lb(0:nc_index*nr_index), stat=ios)
call check_error(ios,'Error allocating lb.',iam)

allocate(lo(nc_dom*nr_dom), stat=ios)
call check_error(ios,'Error allocating lo.',iam)

lb(0) = .false.
gmask = 0.0
ngeos = nc_index*nr_index

```

```

glis = nc_dom*nr_dom
ferror = 1
!-----
! Open GEOS forcing file
!-----
#if ( defined OPENDAP )
  if ( order == 1 ) then
    geos_index = get_geos_index(0)
  else ! order == 2
    geos_index = get_geos_index(3)
  endif
  write(cgeos_index, '(i4)') geos_index

  print*, 'MSG: readgeos -- Retrieving GEOS forcing file ', &
           trim(name), ',(,iam,)'
  call system("opendap_scripts/getgeos.pl "//ciam//" //&
              trim(name)//" "// &
              cgeos_index//" //cgeos_slat//" //cgeos_nlat)
#endif
  print*, 'MSG: readgeos -- Reading GEOS forcing file -', &
           trim(name), ',(,iam,)

  open(40,file=name,form='unformatted',iostat=ios)
  if ( ios /= 0 ) then
    print*, 'ERR: readgeos -- Error opening ', &
             trim(name), '. Stopping.', ',(, iam,)'
    call endrun
  endif

#if ( defined OPENDAP )
  do i = 1,geosdrv%nmif
    read(40,iostat=ioerror)tempvar(:,:,i)
  enddo
#else
  read(40,iostat=ioerror)tempvar
#endif
  if ( ioerror /= 0 ) then
    print*, 'ERR: readgeos -- Error reading ', &
             trim(name), '. Stopping.', ',(, iam,)'
    call endrun
  else
    print*, 'MSG: readgeos -- Read GEOS forcing file -', &
             trim(name), ',(, iam,)'
  endif
!-----
! Finding number of forcing variables
! (13 if time step is 0, otherwise the normal 10)
!-----

```

```

if (tscount .eq. 0) then
    nforce = geosdrv%nmif
else
    nforce = 10 !change later?
endif
do v=1,nforce
!-----
! Transferring current data to 1-D array for interpolation
!-----
c=0
do i=1,nr_index      !i=1,lf%nrold
    do j=1,nc_index !j=1,lf%ncold
        c = c + 1
        f(c) = tempvar(j,i,v)
        if (tscount .eq. 0 .and. order .eq. 1 &
            .and. v .eq. 11) then
            gmask(c) = f(c) ! Storing geos land mask for later use
        endif
    enddo
enddo
!-----
! Initializing input and output grid arrays
!-----
kgdso = 0
kgdso = ld%kgds
if (v .eq. 8 .or. v .eq. 9) then
    ibi = 1
else
    ibi = 1
endif
!-----
! Defining input data bitmap
!-----
do i=1,ngeos
    lb(i)=.true.
enddo
!-----
! Alter default bitmap prescribed above for
! surface parameters (soil wetness, snow)
!-----
if (v .eq. 12 .or. v .eq. 13) then
    do i=1,ngeos
        if (gmask(i)==100.0 .or. gmask(i)==101.0) then
            lb(i)=.false.
        else
            lb(i)=.true.
        endif
    enddo

```

```

        endif
!-----
! Defining output data bitmap
!-----
    do i=1,glis
        lo(i)=.true.
    enddo
!-----
! Interpolate data from GEOS grid to GLDAS grid
!-----
!      if(v.eq.8 .or. v.eq. 9) then
!          CALL POLATES3(kgdso,ibi,lb,f,ibo,lo,go,mi, &
!                         rlat3,rlon3,w113,w123,w213,w223,n113,n123,n213,n223,iret)
!      else
!          CALL POLATES0(kgdso,ibi,lb,f,ibo,lo,go,mi, &
!                         rlat0,rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
!      endif
!-----
! Convert data to original 3D array & a 2D array to
! fill in of missing points due to geography difference
!-----
    count = 0
    do j = 1+nroffset, nr_dom+nroffset !j = 1, ld%nr
        do i = 1, nc_dom           !i = 1, ld%nc
            if(gindex(i,j) .ne. -1) then
                tempgeos(gindex(i,j),v) = go(i+count)
            endif
        enddo
        count = count + ld%lnc
    enddo
!-----
! Fill in undefined and ocean points
!-----
    do i = 1, ld%ngrid
        if (tempgeos(i,v) >= 9.9e+14) then
            tempgeos(i,v) = ld%udef
        endif
        if(order.eq.1)then
            glbdata1(v,i)=tempgeos(i,v)
        else
            glbdata2(v,i)=tempgeos(i,v)
        endif
    enddo           !i
enddo             !v

print*, 'DBG: readgeos -- Deallocating arrays'

deallocate(tempvar, stat=ios)

```

```

call check_error(ios,'Error deallocating tempvar.',iam)

deallocate(tempgeos, stat=ios)
call check_error(ios,'Error deallocating tempgeos.',iam)

deallocate(f, stat=ios)
call check_error(ios,'Error deallocating f.',iam)

deallocate(go, stat=ios)
call check_error(ios,'Error deallocating go.',iam)

deallocate(gmask, stat=ios)
call check_error(ios,'Error deallocating gmask.',iam)

deallocate(lb, stat=ios)
call check_error(ios,'Error deallocating lb.',iam)

deallocate(lo, stat=ios)
call check_error(ios,'Error deallocating lo.',iam)
call absoft_release_cache()

print*, 'MSG: readgeos -- Closing GEOS forcing file -', &
        trim(name), ' (',iam, ')'
close(40, iostat=ios)
if ( ios /= 0 ) then
    print*, 'ERR: readgeos -- Error closing ', trim(name), &
        '. Stopping.', ' (', iam, ')'
    call endrun
endif

print*, 'DBG: readgeos -- leaving', ' (', iam, ')'

return

```

1.24.1 check_error (Source File: *readgeos.F90*)

Error check; Program exits in case of error

INTERFACE:

```
subroutine check_error(ierr,msg,iam)
```

CONTENTS:

```

if ( ierr /= 0 ) then
    print*, 'ERR: readgeos -- ',msg,' Stopping.', ' (',iam, ')'
    call endrun

```

```
endif
```

1.24.2 time_interp_geos.F90 (Source File: time_interp_geos.F90)

Opens, reads, and interpolates GEOS forcing.

TIME1 = most recent past data

TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.25 Core Functions of time_interp_geos

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
1 Oct 1999: Jared Entin; Initial code
25 Oct 1999: Jared Entin; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HFILE
27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave & longwave data
              Altered times which are passed into ZTERP--used to be GMT1
              and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
subroutine time_interp_geos()
```

USES:

```
use lisdrv_module, only : lis, grid
use baseforcing_module, only : glbdata1, glbdata2
use time_manager
use grid_spmdMod
use time_module, only : time2date
use spmdMod
use geosdomain_module, only : geosdrv
```

CONTENTS:

```

if(masterproc) then
  if (get_nstep() .eq. 0) then
    lis%f%nforce = lis%f%nmif
  else
    lis%f%nforce = lis%f%nf
  endif
endif
#endif(defined SPMD)
call MPI_BCAST(geosdrv%geostime1,1,MPI_REAL8,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(geosdrv%geostime2,1,MPI_REAL8,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(lis%t%time,1,MPI_REAL8,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(lis%f%nforce,1,MPI_INTEGER,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(lis%f%shortflag,1,MPI_INTEGER,0, &
  MPI_COMM_WORLD,ier)
call MPI_BCAST(lis%f%longflag,1,MPI_INTEGER,0, &
  MPI_COMM_WORLD,ier)
#endif
btme=geosdrv%geostime1
call time2date(btme,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
btme=geosdrv%geostime2
call time2date(btme,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
!-----
! Interpolate Data in Time
!-----
wt1=(geosdrv%geostime2-lis%t%time)/ &
  (geosdrv%geostime2-geosdrv%geostime1)
wt2=1.0-wt1

do f=1,lis%f%nforce
  if(f.eq.3) then
    if (lis%f%shortflag.eq.2) then
!-----
! Got Time Averaged SW
!-----
    do c=1,gdi(iam)
      zdoy=lis%t%doy
      call zterp(0,grid(c)%lat,grid(c)%lon, &
        gmt1,gmt2,lis%t%gmt,zdoy, &
        zw1,zw2,czb,cze,czm,lis)
      grid(c)%forcing(f)=glbdata2(f,c)*zw1
      if ((grid(c)%forcing(f).ne.lis%d%udef).and. &

```

```

        (grid(c)%forcing(f).lt.0) ) then
        if (grid(c)%forcing(f) > -0.00001) then
            grid(c)%forcing(f) = 0.0
        else
            print*, 'ERR: time_interp_geos -- Stopping because ', &
                  'forcing not udef but lt0,'
            print*, 'ERR: time_interp_geos -- ', &
                  'f,c,grid(c)%forcing(f),glbdata2(f,c)',  &
                  f,c,grid(c)%forcing(f),glbdata2(f,c),  &
                  ',(,iam,)'
            call endrun
        end if
    endif

    if (grid(c)%forcing(f).gt.1367) then
        grid(c)%forcing(f)=glbdata2(f,c)
    endif
enddo
endif

else if(f.eq.8.or.f.eq.9) then
!---
! precip variable Block Interpolation
!---
    do c=1,gdi(iam)
        grid(c)%forcing(f)=glbdata2(f,c)
    enddo

    else if (f.eq.4) then
        if (lis%f%longflag.eq.1) then
!---
!      Got Instantaneous LW
!---
        do c=1,gdi(iam)
            grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
                glbdata2(f,c)*wt2
        enddo
    endif

    if (lis%f%longflag.eq.2) then
!---
!      Got Time Averaged LW
!---
        do c=1,gdi(iam)
            grid(c)%forcing(f)=glbdata2(f,c)

        enddo
    endif

```

```

        else
!-----
!      Linearly interpolate everything else
!-----
        do c=1,gdi(iam)
          grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
            glbdata2(f,c)*wt2
        enddo
      endif
    enddo
84  format('now',i4,4i3,2x,'pvt ',a22,' nxt ',a22)
    return

```

1.26 Fortran: Module Interface gdasdomain_module.F90 (Source File: gdasdomain_module.F90)

Contains routines and variables that define the native domain for GDAS model forcing

INTERFACE:

```
module gdasdomain_module
```

USES:

```
use gdasdrv_module
```

1.26.1 defnatgdas.F90 (Source File: gdasdomain_module.F90)

Defines the kgds array describing the native forcing resolution for GDAS data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnatgdas(kgdsi)
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

ARGUMENTS:

```
integer, intent(inout) :: kgdsi(200)
```

CONTENTS:

```

call readgdascrd(gdasdrv)
kgdsi(1) = 4
kgdsi(2) = 512
kgdsi(3) = 256
kgdsi(4) = 89463
kgdsi(5) = 0
kgdsi(6) = 128
kgdsi(7) = -89463
kgdsi(8) = -703
kgdsi(9) = 703
kgdsi(10) = 128
kgdsi(20) = 255
mi = gdasdrv%ncold*gdasdrv%nrold

```

1.27 Fortran: Module Interface gdasdrv_module.F90 (Source File: gdasdrv_module.F90)

Module containing runtime specific GDAS variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module gdasdrv_module
```

ARGUMENTS:

```

type gdasdrvdec
    integer :: ncold, nrold    !AWIPS 212 dimensions
    integer :: nmif
    character*40 :: gdasdir   !GDAS Forcing Directory
    real*8 :: gdastime1, gdastime2
end type gdasdrvdec

```

1.27.1 getgdas.F90 (Source File: getgdas.F90)

Opens, reads, and interpolates NCEP-GDAS forcing.

TIME1 = most recent past data

TIME2 = nearest future data

The idea is to open either the 00 or 03 forecast file associated with the most recent GDAS assimilation (available every 6 hours). Precipitation rates and radiation fluxes will be taken

from the F03 and F06 files, since averages are provided.

- if that fails, the strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.28 Core Functions of getgdas

tick Determines GDAS data times

gdasfile Puts together appropriate file name for 3 hour intervals

gdasfilef06 Puts together appropriate file name for 6 hour intervals

retgdas Interpolates GDAS data to LDAS grid

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
25 Oct 1999: Jared Entin; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HRFIL
27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave and longwave data.
              Altered times which are passed into ZTERP--used to be GMT1 and GMT2,
              now they are LDAS%ETATIME1 and LDAS%ETATIME2
11 May 2000: Brian Cosgrove; Added checks for SW values that are too high
              due to zenith angle weighting...if too high, use linear weighting.
              Also, if cos(zen) less than .1, then use linear weighting to
              avoid computed values of SW that are too high.
18 May 2000: Brian Cosgrove; Corrected line of code in ETAEDASNAME which
              assigned wrong year directory variable when constructing
              EDAS filename
26 May 2000: Jared Entin; Changed numerical bound of the TRY variable
              to fix a rollback problem.
5 June 2000: Brian Cosgrove; Fixed a problem with the correction of the negative
              radiation sign convention. Prior to fix, was not correcting negative
              values of -999.9...now it changes all negative values to positive ones.
18 Aug 2000: Brian Cosgrove; Fixed undefined value problem in check for negative
              radiation values over land points.
08 Dec 2000: Urszula Jambor; Rewrote geteta.f in fortran90 to use GDAS in GLDAS
15 Mar 2001: Jon Gottschalck; Slight change to handle more forcing parameters at
              time step 0.
09 Apr 2001: Urszula Jambor; Added capability of using DAAC forcing data every
              6 hours, rather than every 3 hours.
30 May 2001: Urszula Jambor; Changed forcing used: T,q,u fields taken
              from F00 & F03 files, radiation and precip. fields taken
              from F06 & F03 (F03 fields are subtracted out from F06)

```

INTERFACE:

```
subroutine getgdas()
```

USES:

```
use lisdrv_module, only : lis, gindex
use baseforcing_module, only: glbdata1, glbdata2
use time_manager
use time_module, only : tick
use gdasdomain_module, only : gdasdrv
```

CONTENTS:

```
lis%f%FOO_flag = 0
lis%f%F06_flag = 0
lis%f%findtime1=0
lis%f%findtime2=0
movetime=0
!-----
! Determine the correct number of forcing variables
!-----
if(get_nstep().eq.0 ) then
    nforce = gdasdrv%nmif
else
    nforce = lis%f%nf
endif
if (get_nstep() .eq. 1.or.lis%f%rstflag.eq.1) then
    lis%f%findtime1=1
    lis%f%findtime2=1
    glbdata1 = 0
    glbdata2 = 0
    movetime=0
    lis%f%rstflag = 0
endif
!-----
! Determine required GDAS data times
! (previous assimilation, current & future assimilation hours)
! The adjustment of the hour and the direction will be done
! in the subroutines that generate the names
!-----
yr1 = lis%t%yr !current time
mo1 = lis%t%mo
da1 = lis%t%da
hr1 = lis%t%hr
mn1 = lis%t%mn
ss1 = 0
ts1 = 0
call tick( timenow, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
yr1 = lis%t%yr !previous assimilation/forecast hour
```

```

mo1 = lis%t%mo
da1 = lis%t%da
if ( lis%f%F06_flag == 0 ) then
    hr1 = 3*(int(real(lis%t%hr)/3.0))
else
    hr1 = 6*(int(real(lis%t%hr)/6.))
end if
mn1 = 0
ss1 = 0
ts1 = 0
call tick( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

yr2 = lis%t%yr !next assimilation/forecast hour
mo2 = lis%t%mo
da2 = lis%t%da
if ( lis%f%F06_flag == 0 ) then
    hr2 = 3*(int(real(lis%t%hr)/3.0))
else
    hr2 = 6*(int(real(lis%t%hr)/6.0))
end if
mn2 = 0
ss2 = 0
if ( lis%f%F06_flag == 0 ) then
    ts2 = 3*60*60
else
    ts2 = 6*60*60
end if
call tick( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
!-----
! Use these if need to roll back time.
!-----
dumbtime1 = time1
dumbtime2 = time2
!-----
! Check to see if current time (timenow) has crossed past gdastime2,
! requiring that both gdastime2 be assigned to gdastime1 and a new
! gdastime2 be set 3 or 6 hours ahead of the current gdastime2.
!-----
if ( timenow > gdasdrv%gdastime2 ) then
    movetime = 1
    lis%f%findtime2 = 1
end if
!-----
! Establish gdastime1
!-----
if ( lis%f%findtime1 == 1 ) then !get new time1 from the past
    print *, 'Getting new time1 data'
    ferror = 0

```

```

order = 1
try = 0
ts1 = -24*60*60

do
  if ( ferror /= 0 ) exit
  try = try+1
  if ( lis%f%F06_flag == 0 ) then
    call gdasfile  ( name, gdasdrv%gdasdir, yr1, mo1, da1, hr1 )
  else
    call gdasfileF06( name, gdasdrv%gdasdir, yr1, mo1, da1, hr1 )
  end if
  print*, 'Reading GDAS file ',name
  call retgdas( order, lis, gindex, name, nameF06, 0,ferror, try)

  if ( ferror == 1 ) then
!---
! successfully retrieved forcing data
!---
      gdasdrv%gdastime1 = time1
      else
!---
! ferror still=0, so roll back one day & start again
!---
      call tick( dumptime1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
      end if
      if ( try > ndays ) then
!---
! data gap exceeds 10 days so stop
!---
      print *, 'ERROR: GDAS data gap exceeds 10 days on file 1'
      call endrun
      end if
    end do
  end if
!---
! Establish gdastime2
!---
  if ( movetime == 1 ) then
    gdasdrv%gdastime1 = gdasdrv%gdastime2
    lis%f%findtime2 = 1
    do f = 1, nforce
      do c = 1, lis%d%ngrid
        glbdata1(f,c) = glbdata2(f,c)
      end do
    end do
  end if

```

```

if ( lis%f%findtime2 == 1 ) then
    print *, 'Getting new time2 data'
    ferror = 0
    order = 2
    try = 0
    ts2 = -24*60*60
!-----
! determine if both F00 & F06 files needed
!-----
if ( lis%f%F06_flag == 0 ) then
    if ( modulo(hr2,2) > 0 ) then
        lis%f%F00_flag = 0 !odd hr, use F03 file
    else
        lis%f%F00_flag = 1 !even hr, use F00 & F06
    end if
end if

do
    if ( ferror /= 0 ) exit
    try = try+1
    if ( lis%f%F06_flag == 0 ) then
        if ( lis%f%F00_flag == 0 ) then
            call gdasfile( name,gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
        else
            call gdasfile( name,gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
            call gdasfileF06( nameF06, gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
        end if
    else
        call gdasfileF06( name, gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
    end if
    print*, 'Reading GDAS file ',name
    call retgdas( order, lis, gindex, name, nameF06, &
        lis%f%F00_flag, ferror,try )
    if ( ferror == 1 ) then
!-----
! successfully retrieved forcing data
!-----
        gdasdrv%gdastime2 = time2
    else
!-----
! ferror still=0, so roll back one day & start again
!-----
        call tick( dumbtime2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
    end if
    if ( try > ndays ) then
!-----
! data gap exceeds 10 days so stop
!-----

```

```

        print *, 'ERROR: GDAS data gap exceeds 10 days on file 2'
        call endrun
    end if
end do
end if
!-----
! loop through all forcing parameters & interpolate
!-----
do f = 1, lis%f%nforce
    if ( (f == 3) .or. (f == 4) ) then
        do c = 1, lis%d%lnc
            do r = 1, lis%d%lnr
                index = gindex(c,r)
                if(index .ne.-1) then
                    if ( (glbdata2(f,index) /= -9999.9) .and.  &
                        (glbdata2(f,index) < 0)) then
                        glbdata2(f,index) = (-1) * glbdata2(f,index)
                    end if
                    if ( (glbdata1(f,index) /= -9999.9) .and.  &
                        (glbdata1(f,index) < 0)) then
                        glbdata1(f,index) = (-1) * glbdata1(f,index)
                    end if
                end if
            end do
        end do
    end if
enddo

```

1.28.1 gdasfile (Source File: getgdas.F90)

This subroutine puts together GDAS file name for 3 hour file intervals

INTERFACE:

```

subroutine gdasfile( name, gdasdir, yr, mo, da, hr )

implicit none

```

INPUT PARAMETERS:

```

character(len=80) :: gdasdir
integer :: yr, mo, da, hr

```

OUTPUT PARAMETERS:

```

character(len=80) :: name

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, a2)
95 format (10a1)
96 format (a40)
97 format (a16, a2, a3)
98 format (a1, i4, i2, a1)
99 format (8a1)
!-----
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----
uyr = yr
umo = mo
uda = da
uhr = 3*(hr/3) !hour needs to be a multiple of 3 hours
umn = 0
uss = 0
ts1 = -24*60*60 !one day interval to roll back date.
remainder = modulo(uhr,2) !if even, then remainder equals zero
                           !if odd, then remainder equals one
!-----
! if hour is    00 or 03, look for 00ZF00 or 00ZF03
! if hour is    06 or 09, look for 06ZF00 or 06ZF03
! if hour is    12 or 15, look for 12ZF00 or 12ZF03
! if hour is    18 or 21, look for 18ZF00 or 18ZF03
!-----
if ( uhr <= 3 ) then
  initcode = '00'
else if (uhr <= 9 ) then
  initcode = '06'
else if ( uhr <= 15 ) then
  initcode = '12'
else if ( uhr <= 21 ) then
  initcode = '18'
end if
if ( remainder > 0 ) then
  fcstcode = '03'
else
  fcstcode = '00'
end if

write(UNIT=temp, fmt='(a40)') gdasdir
read(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,80)

write(UNIT=temp, fmt='(a1, i4, i2, a1)') '/', uyr, umo, '/'

```

```

read(UNIT=temp, fmt='(8a1)') fdir
do i = 1, 8
    if ( fdir(i) == ' ') fdir(i) = '0'
end do

write(UNIT=temp, fmt='(i4, i2, i2, a2)') uyr, umo, uda, initcode
read(UNIT=temp, fmt='(10a1)') ftime
do i = 1, 10
    if ( ftime(i) == ' ') ftime(i) = '0'
end do

write(UNIT=temp, fmt='(a16, a2, a3)') '.gdas1.sfluxgrbf', fcstcode, '.sg'
read (UNIT=temp, fmt='(80a1)') (fsubs(i), i=1,21)

c = 0
do i = 1, 80
    if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,8), &
                               (ftime(i), i=1,10), (fsubs(i), i=1,21)

read(UNIT=temp, fmt='(a80)') name

return

```

1.28.2 gdasfilef06 (Source File: getgdas.F90)

This subroutine puts together GDAS file name for 6 hour forecast files (DAAC archive friendly)

INTERFACE:

```
subroutine gdasfileF06( name, gdasdir, yr, mo, da, hr )
```

USES:

```
use time_module

implicit none
```

INPUT PARAMETERS:

```
character(len=80) :: gdasdir
integer :: yr, mo, da, hr
```

OUTPUT PARAMETERS:

```
character(len=80) :: name
```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, a2)
95 format (10a1)
96 format (a40)
97 format (a16, a2, a3)
98 format (a1, i4, i2, a1)
99 format (8a1)

!-----
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----

uyr = yr
umo = mo
uda = da
uhr = 6*(hr/6) !hour needs to be a multiple of 6 hours
umn = 0
uss = 0
ts1 = -24*60*60 !one day interval to roll back date.

!-----
! if hour is 00Z look for 18ZF06 after rolling back one day
! if hour is 06Z look for 00ZF06
! if hour is 12Z look for 06ZF06
! if hour is 18Z look for 12ZF06
!-----

fcstcode = '06'
if ( uhr == 0 ) then
    call tick( dumbtime, doy, gmt, uyr, umo, uda, uhr, umn, uss, ts1 )
    initcode = '18'
else if ( uhr == 6 ) then
    initcode = '00'
else if ( uhr == 12 ) then
    initcode = '06'
else if ( uhr == 18 ) then
    initcode = '12'
end if

write(UNIT=temp, fmt='(a40)') gdasdir
read(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,80)

write(UNIT=temp, fmt='(a1, i4, i2, a1)') '/', uyr, umo, '/'
read(UNIT=temp, fmt='(8a1)') fdir
do i = 1, 8
    if ( fdir(i) == ' ' ) fdir(i) = '0'
end do

```

```

write(UNIT=temp, fmt='(i4, i2, i2, a2)') uyr, umo, uda, initcode
read(UNIT=temp, fmt='(10a1)') ftime
do i = 1, 10
    if ( ftime(i) == ' ') ftime(i) = '0'
end do
write(UNIT=temp, fmt='(a16, a2, a3)') '.gdas1.sfluxgrbf', '06', '.sg'
read (UNIT=temp, fmt='(80a1)') (fsubs(i), i=1,21)
c = 0
do i = 1, 80
    if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,8), &
                               (ftime(i), i=1,10), (fsubs(i), i=1,21)

read(UNIT=temp, fmt='(a80)') name
return

```

1.28.3 readgdascrd.F90 (Source File: readgdascrd.F90)

Routine to read GDAS specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readgdascrd(gdasdrv)
```

USES:

```
use gdasdrv_module
```

CONTENTS:

```

open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=gdas)
print*, 'Using GDAS forcing'
print*, 'GDAS forcing directory : ',gdasdrv%GDASDIR
gdasdrv%GDASTIME1 = 3000.0
gdasdrv%GDASTIME2 = 0.0

close(11)

```

1.28.4 retgdas.F90 (Source File: retgdas.F90)

Defines forcing parameters, retrieves the fields using calls to getgb, and interpolates the fields to LDAS specifications

REVISION HISTORY:

```

14 Dec 2000: Urszula Jambor; Rewrote geteta.f to use GDAS forcing in GLDAS
15 Mar 2001: Jon Gottschalck; Added additional parameters and octets in
              which to search in GRIB files
01 Jun 2001: Urszula Jambor; Added option to get forcing from different
              files (F00 instantaneous and F06 six hour means)
29 Jan 2003: Urszula Jambor; Rewrote code, uses GETGB call to replace
              ungribgdas. Interpolation now occurs in interp_gdas.
              Using GETGB avoids problems with the Oct2002 GDAS
              grid update.
12 Nov 2003: Matt Rodell; Check to make sure input file exists before
              opening and thereby creating a new, empty file.
14 Nov 2003: Matt Rodell; Ensure lugb varies in call to baopen

```

INTERFACE:

```
subroutine retgdas( order, ld, gindex, name, nameF06, F00flag,ferror,try )
```

USES:

```

use lis_module          ! LDAS non-model-specific 1-D variables
use time_manager
use baseforcing_module, only: glbdata1, glbdata2
use gdasdomain_module, only : gdasdrv
implicit none

```

ARGUMENTS:

```

type (lisdec) ld
integer :: gindex(ld%d%lnc, ld%d%lnr)
integer :: order      ! 1 indicates lesser interp. bdry, 2 indicates greater
character(len=80) :: name, nameF06
integer :: F00flag   ! if 1, need for data from 2 files (name, nameF06)
integer :: ferror    ! set to zero if there's an error
integer :: try

```

CONTENTS:

```

varfield = 0.0
ngdas = (gdasdrv%ncold*gdasdrv%nrold)
!-----
! Set the GRIB parameter specifiers
!-----
if (get_nstep() .eq. 0) then
  pds5 = (/011,051,204,205,033,034,001,059,214,084,144,144, 011,011, 065/) !parameter
  pds7 = (/002,002,000,000,010,010,000,000,010,2760,010,2760,000/) !htlev2

```

```

    nforce = gdasdrv%nmif
else
  pds5 = (/ 011,051,204,205,033,034,001,059,214,084 /) !parameter
  pds7 = (/ 002,002,000,000,010,010,000,000,000,000 /) !htlev2
  nforce = 10 ! for now
endif

ferror = 1
!-----
! if there's a problem then ferror is set to zero
!-----
iv = 0
errorflag = 0
endloop = 0

do
  if ( endloop == 1 ) exit
  iv = iv+1

  if ( (F00flag > 0) .and. &
       (iv==3.or.iv==4.or.iv==8.or.iv==9.or.iv==10) ) then
    fname = nameF06
  else
    fname = name
  end if
  inquire (file=fname, exist=file_exists)
  if (file_exists) then
!-----
! Set up to open file and retrieve specified field
!-----
    lugb = iv +try
    j = 0
    jpds = -1
    jpds(5) = pds5(iv)
    jpds(7) = pds7(iv)
    call baopen(lugb,fname,iret)
    if(iret==0) then
      call getgb(lugb,lubi,ngdas,j,jpds,jgds,kf,k,kpds,kgds,lb,f,gbret)
    else
      gbret = 99
    endif
    call baclose(lugb,jret)
  else
    ferror = 0
    return
  endif
!-----
! If field successfully retrieved, interpolate to LIS domain

```

```
!-----
      if (gbret==0) then
          call interp_gdas(kpds,kgds,ngdas,f,lb,ld%d%kgds, &
                           ld%d%lnc,ld%d%lnr,varfield)
      else
          errorflag = 1
      endif

      if ( errorflag == 1 ) then
          endloop = 1
          ferror = 0
      else
          do c =1, ld%d%lnc
              do r = 1, ld%d%lnr
                  if (gindex(c,r).ne. -1) then
                      if ( order == 1 ) then
                          glbdata1(iv,gindex(c,r)) = varfield(c,r)
                      else
                          glbdata2(iv,gindex(c,r)) = varfield(c,r)
                      end if
                  endif
              end do
          end do
      end if

      if ( errorflag == 1 ) then
          print *, 'Could not find correct forcing parameter in file',name
      end if
      if ( iv == nforce ) endloop = 1
  end do
  return
```

1.28.5 interp_gdas (Source File: retgdas.F90)

This subroutine interpolates a given GDAS field to the LIS domain. Special treatment for some initialization fields. Code based on old ungribgdas.f

INTERFACE:

```
subroutine interp_gdas(kpds,kgds,ngdas,f,lb,lis_gds,nc,nr, &
                       varfield)
```

USES:

```
use def_ipMod, only : w110,w120,w210,w220,n110,n120,n210,n220,&
                     rlat0,rlon0
use gdasdomain_module, only : mi
```

CONTENTS:

```

!-----
! Setting interpolation options (ip=0,bilinear)
! (km=1, one parameter, ibi=1,use undefined bitmap
! (needed for soil moisture and temperature only)
! Use budget bilinear (ip=3) for precip forcing fields
!-----

nglis = nc*nr
if (kpds(5)==59 .or. kpds(5)==214) then
    ip=3
    ipopt(1)=-1
    ipopt(2)=-1
    km=1
    ibi=1
else
    ip=0
    do i=1,20
        ipopt(i)=0
    enddo
    km=1
    ibi=1
endif
!-----
! Initialize output bitmap. Important for soil moisture and temp.
!-----
lo = .true.
!-----
! Interpolate to LIS grid
!-----
call polates0 (lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,&
               rlat0, rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
!-----
! Create 2D array for main program. Also define a "soil" mask
! due to different geography between GDAS & LDAS. For LDAS land
! points not included in GDAS geography dataset only.
!-----
count = 0
do j = 1, nr
    do i = 1, nc
        varfield(i,j) = lis1d(i+count)
        geogmask(i,j) = lo(i+count)
    enddo
    count = count + nc
enddo
!-----
! Save air tempertaure interpolated field for later use in
! initialization of soil temp where geography differs

```

```

! between GDAS and LDAS
!-----
if (kpds(5) .eq. 11 .and. kpds(6) .eq. 105) then
  do i = 1, nc
    do j = 1, nr
      geogtemp(i,j) = varfield(i,j)
    enddo
  enddo
endif

```

1.28.6 time_interp_gdas (Source File: time_interp_gdas.F90)

Opens, reads, and interpolates GDAS forcing.

TIME1 = most recent past data

TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.29 Core Functions of time_interp_gdas

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
25 Oct 1999: Jared Entin; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HFILE
27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave & longwave data
              Altered times which are passed into ZTERP--used to be GMT1
              and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.

```

INTERFACE:

```
subroutine time_interp_gdas()
```

USES:

```

use lisdrv_module, only :lis, grid
use baseforcing_module, only: glbdata1, glbdata2

```

```

use time_manager
use time_module, only : tick, time2date
use grid_spmdMod
use spmdMod
use gdasdomain_module, only : gdasdrv

```

CONTENTS:

```

if(masterproc) then
  if (get_nstep() .eq. 0) then
    lis%f%nforce = gdasdrv%nmif
  else
    lis%f%nforce = lis%f%nf
  endif
endif
#endif SPMD)
call MPI_BCAST(gdasdrv%gdastime1,1,MPI_REAL8,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(gdasdrv%gdastime2,1,MPI_REAL8,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(lis%t%time,1,MPI_REAL8,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(lis%f%nforce,1,MPI_INTEGER,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(lis%f%FOO_flag,1,MPI_INTEGER,0, &
               MPI_COMM_WORLD,ierr)
call MPI_BCAST(lis%f%F06_flag,1,MPI_INTEGER,0, &
               MPI_COMM_WORLD,ierr)

#endif
btime=gdasdrv%gdastime1
call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
btime=gdasdrv%gdastime2
call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
if ( (lis%f%F06_flag==0) .and. (lis%f%FOO_flag==1) ) then
  inittime = gdasdrv%gdastime2
  call time2date( inittime, idoy, igmt, iyr, imo, ida, ihr, imn )
  its = -6*60*60
  call tick( inittime, idoy, igmt,iyr,imo,ida,ihr,imn,iss,its)
end if
wt1 = (gdasdrv%gdastime2-lis%t%time) / &
      (gdasdrv%gdastime2-gdasdrv%gdastime1)
wt2 = 1.0 - wt1
do f=1,lis%f%nforce
  if ( f == 3 ) then !shortwave
    do c = 1, gdi(iam)
      zdoy = lis%t%doy
      if ( (lis%f%F06_flag==0) .and. (lis%f%FOO_flag==1) ) then

```

```

    call zterp( 0, grid(c)%lat, grid(c)%lon, igmt, gmt2, &
                lis%t%gmt,zdoy,zw1,zw2,czb,cze,czm,lis)
  else
    call zterp( 0, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
                lis%t%gmt,zdoy,zw1,zw2,czb,cze,czm,lis)
  end if
  grid(c)%forcing(f) = zw1 * glbdata2(f,c)
  if (grid(c)%forcing(f) < 0) then
    print *, '2 warning!!! SW radiation is negative!!'
    print *, 'sw=', grid(c)%forcing(f), '... negative'
    print *, 'gdas2=', glbdata2(f,c)
    call endrun
  end if

  if (grid(c)%forcing(f).gt.1367) then
    grid(c)%forcing(f)=glbdata2(f,c)
  endif
  end do
else if ( (f==4) .or. (f==10) ) then
  do c = 1, gdi(iam)
    if ( lis%f%F00_flag==1 ) then
      grid(c)%forcing(f)=2*glbdata2(f,c) -glbdata1(f,c)
    else
      grid(c)%forcing(f) = glbdata2(f,c)
    end if
  end do

else if ( (f==8) .or. (f==9) ) then
  do c = 1, gdi(iam)
    if ( lis%f%F00_flag == 1) then
      if (2*glbdata2(f,c) >= glbdata1(f,c)) then
        grid(c)%forcing(f)=2*glbdata2(f,c)-glbdata1(f,c)
      else
        grid(c)%forcing(f) = 0.0
      end if
    else
      grid(c)%forcing(f) = glbdata2(f,c)
    end if
  end do
else
  do c = 1, gdi(iam)
    grid(c)%forcing(f) = wt1 * glbdata1(f,c) +  &
                           wt2 *glbdata2(f,c)
  end do
end if
end do
return

```

1.30 Fortran: Module Interface baseforcing_pluginMod.F90 (Source File: baseforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

11 Dec 03 Sujay Kumar Initial Specification

INTERFACE:

```
module baseforcing_pluginMod
```

1.30.1 baseforcing_plugin (Source File: baseforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine baseforcing_plugin
```

USES:

```
use geosdomain_module
use gdasdomain_module
```

CONTENTS:

```
call registerget(1,getgdas)
call registerget(2,getgeos)

call registerdefnat(1,defnatgdas)
call registerdefnat(2,defnatgeos)

call registertimeinterp(1,time_interp_gdas)
call registertimeinterp(2,time_interp_geos)
```

1.31 Fortran: Module Interface precipforcing_pluginMod.F90 (Source File: precipforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new observed precipitation forcing scheme.

REVISION HISTORY:

12 Dec 03 Sujay Kumar Initial Specification

INTERFACE:

```
module precipforcing_pluginMod
```

1.31.1 precipforcing_plugin (Source File: precipforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new observed precipitation forcing scheme. The interface mandates that the following routines be implemented and registered for each of the forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnatpcp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine precipforcing_plugin
```

USES:

```
use huffdomain_module
use persdomain_module
use cmapdomain_module
```

CONTENTS:

```
call registerpget(2, gethuff)
call registerpget(3, getpers)
call registerpget(4,getcmap)

call registerdefnatpcp(2, defnathuff)
call registerdefnatpcp(3, defnatpers)
call registerdefnatpcp(4, defnatcmap)
```

```

call registerpti(2,time_interp_huff)
call registerpti(3,time_interp_pers)
call registerpti(4,time_interp_cmap)

end subroutine precipforcing_plugin
end module precipforcing_pluginMod

\markboth{Left}{Source File: radforcing\_pluginMod.F90, Date: Mon Dec 22 15:57:48 EST 2003}

!-----
!      NASA GSFC Land Information Systems LIS 2.3
!-----

```

1.32 Fortran: Module Interface radforcing_pluginMod.F90 (Source File: radforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new observed radiation forcing scheme.

REVISION HISTORY:

12 Dec 03 Sujay Kumar Initial Specification

INTERFACE:

```
module radforcing_pluginMod
```

1.32.1 radforcing_plugin (Source File: radforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new observed radiation forcing scheme. The interface mandates that the following routines be implemented and registered for each of the forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnatrad)

Temporal interpolation Routines to temporally interpolate data (to be registered using registerrti)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine radforcing_plugin
```

USES:

```
use agrmetdomain_module
```

CONTENTS:

```
call registererrget(1,getgrad)
call registerdefnatrad(1,defnatagrm)
call registerrti(1,time_interp_agrmet)
```

1.33 Fortran: Module Interface *lsm_pluginMod.F90* (Source File: *lsm_pluginMod.F90*)

This module contains the definition of the functions used for land surface model initialization, execution, reading and writing of restart files and other relevant land surface model computations, corresponding to each of the LSMs used in LIS.

REVISION HISTORY:

09 Oct 03 Sujay Kumar Initial Specification

INTERFACE:

```
module lsm_pluginMod
```

1.33.1 *lsm_plugin* (Source File: *lsm_pluginMod.F90*)

This is a custom-defined plugin point for introducing a new LSM. The interface mandates that the following routines be implemented and registered for each of the LSM that is included in LIS.

Initialization Definition of LSM variables (to be registered using registerlsmini)

Setup Initialization of parameters (to be registered using registerlsmsetup)

DynamicSetup Routines to setup time dependent parameters (to be registered using registerlsmdynsetup)

Run Routines to execute LSM on a single gridcell for single timestep (to be registered using registerlsmrun)

Read restart Routines to read a restart file for an LSM run (to be registered using registerlsmrestart)

Output Routines to write output (to be registered using registerlsmoutput)

Forcing transfer to model tiles Routines to transfer an array of given forcing to model tiles (to be registered using registerlsmf2t)

Write restart Routines to write a restart file (to be registered using registerlsmwrst)

Multiple LSMs can be included as well, each distinguished in the function table registry by the associated LSM index assigned in the card file.

INTERFACE:

```
subroutine lsm_plugin
```

USES:

```
use noah_varder, only : noah_varder_ini
use clm_varder, only : clm_varder_ini
use vic_varder, only : vic_varder_ini
use atmdrvMod, only : atmdrv
use mos_varder, only : mos_varder_ini
```

CONTENTS:

```
call registerlsmini(1,noah_varder_ini)
call registerlsmini(2,clm_varder_ini)
call registerlsmini(3,vic_varder_ini)
call registerlsmini(4,mos_varder_ini)

call registerlsmsetup(1,noah_setup)
call registerlsmsetup(2,clm2_setup)
call registerlsmsetup(3,vic_setup)
call registerlsmsetup(4, mos_setup)

call registerlsmdynsetup(1,noah_dynsetup)
call registerlsmdynsetup(2,clm2_dynsetup)
call registerlsmdynsetup(3,vic_dynsetup)
call registerlsmdynsetup(4, mos_dynsetup)

call registerlsmrun(1,noah_main)
call registerlsmrun(2,driver)
call registerlsmrun(3,vic_main)
call registerlsmrun(4, mos_main)

call registerlsmrestart(1,noahrst)
call registerlsmrestart(2,clm2_restart)
call registerlsmrestart(3,vic_readrestart)
call registerlsmrestart(4, mosrst)

call registerlsmoutput(1,noah_output)
call registerlsmoutput(2,clm2_output)
call registerlsmoutput(3,vic_output)
call registerlsmoutput(4, mos_output)

call registerlsmf2t(1,noah_f2t)
```

```

call registerlsmf2t(2,atmdrv)
call registerlsmf2t(3,vic_atmdrv)
call registerlsmf2t(4, mos_f2t)

call registerlsmwrst(1,noah_writerst)
call registerlsmwrst(2,clm2wrst)
call registerlsmwrst(3,vic_writerestart)
call registerlsmwrst(4, mos_writerst)

```

ROUTINE: canhtset.F90

This subroutine reads in canopy height information into CLM2 for now

REVISION HISTORY:

15 Nov 2002: Jon Gottschalck; Initial code

INTERFACE:

```
subroutine canhtset ()
```

USES:

```

use clm_varder          ! CLM2 tile variables
use clm_varpar          , only : maxpatch
use clm_varmap          , only : numpatch
use clm_varctl          , only : clmdrv
use lis_module

```

CONTENTS:

```

!-----
! Open canopy heights file and read into temporary variables
!-----
open(57,file=clmdrv%clm2_chtfile,status='old')
read(57,*) (pthtop(i),i=1,maxpatch)
read(57,*) (pthbot(i),i=1,maxpatch)
close(57)
!-----
! Assign canopy top and height values for each
! vegetation type to CLM2 tiles
!-----
do t=1,numpatch
  clm(t)%htop = pthtop(clm(t)%itypveg)
  clm(t)%hbot = pthbot(clm(t)%itypveg)
enddo
return

```

1.33.2 **clm2_almaout.F90** (Source File: *clm2_almaout.F90*)

CLM output writer.

REVISION HISTORY:

```

29 Oct. 1999: Jon Radakovich; Initial code
27 Sep. 2000: Brian Cosgrove; Major revisions to enable CLM to
               output ALMA/LDAS variables
27 Sep. 2000: Added arbitrary root zone cutoff value of .05 so
               that root zone is considered only those levels
               with a rooting value >= .05
19 Mar 2000: Cosgrove; Changed code to allow LDAS GRIB output. Removed
               calls to LATS4D that had been used for grib output and
               added a call to griboutclm which outputs GRIB CLM data.
05 Apr 2002: Jon Gottschalck; Modified code to work with CLM2
14 Jun 2003; Sujay Kumar; ALMA version of the output routine

```

INTERFACE:

```
subroutine clm2_almaout (ld, tile)
```

USES:

```

use lis_module
use tile_module
use clm_varctl, only : clmdrv

```

CONTENTS:

```

!-----
! Test to see if output writing interval has been reached
!-----
if(mod(ld%t%gmt,clmdrv%writeintc2).eq.0)then
!-----
! Generate directory structure and file names for CLM Output
!-----
      write(unit=temp,fmt='(i4,i2,i2)')ld%t%yr,ld%t%mo,ld%t%da
      read(unit=temp,fmt='(8a1)')ftime
      do i=1,8
         if(ftime(i).eq.(' '))ftime(i)='0'
      enddo

      write(unit=temp,fmt='(i4)')ld%t%yr
      read(unit=temp,fmt='(8a1)')ftimec
      do i=1,4
         if(ftimec(i).eq.(' '))ftimec(i)='0'
      enddo

      write(unit=temp,fmt='(a6,i3,a1)')' /LIS.E',ld%o%expcode,'.

```

```

read(unit=temp,fmt='(80a1)') (fname(i),i=1,11)
do i=1,11
    if(fname(i).eq.(' '))fname(i)='0'
enddo

write(unit=temp,fmt='(a40)') ld%o%odir
read(unit=temp,fmt='(40a1)') (fbase(i),i=1,40)
c=0
do i=1,40
    if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
enddo

write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)')'/EXP', &
    ld%o%expcode,'/CLM2/', &
    ld%t%yr,'/',ld%t%yr,ld%t%mo,ld%t%da
read(unit=temp,fmt='(80a1)') (fyrmadir(i),i=1,26)
do i=1,26
    if(fyrmadir(i).eq.(' '))fyrmadir(i)='0'
enddo

write(unit=temp,fmt='(a9)')'mkdir -p '
read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)

write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
    (fyrmadir(i),i=1,26)
read(unit=temp,fmt='(a80)')mkfyrmo

!-----
! Make the directories for the CLM2 output files
!-----
call system(mkfyrmo)
!-----
! Generate file name for binary output
!-----
if(ld%o%wout.eq.1)then
    write(unit=temp,fmt='(I4,I2,I2,I2)')ld%t%yr, &
        ld%t%mo,ld%t%da,ld%t%hr
    read(unit=temp,fmt='(10A1)')ftimeb
    do i=1,10
        if(ftimeb(i).eq.(' '))ftimeb(i)='0'
    enddo

    write(unit=temp,fmt='(A9)')'.CLM2gbin'
    read(unit=temp,fmt='(80A1)') (fsubgb(i),i=1,9)

    write(unit=temp,fmt='(80A1)')(fbase(i),i=1,c), &
        (FYRMODIR(I),I=1,26), &
        (fname(i),i=1,11), &

```

```

        (ftimeb(i),i=1,10),(fsubgb(i),i=1,9 )
      read(unit=temp,fmt='(A80)')filengb
    endif
    if(ld%o%wout.eq.1)then
      open(57,file=filengb,form='unformatted')

      clmdrv%numoutc2=clmdrv%numoutc2+1      !Counts number of output times
!-----
! Write statistical output
!-----
      if(clmdrv%clm2open.eq.0)then
        file='CLMstats.dat'
        call openfile(name,ld%o%odir,ld%o%expcode,file)
        if(ld%o%startcode.eq.1)then
          open(60,file=name,form='formatted',status='unknown', &
            position='append')
        else
          open(60,file=name,form='formatted',status='replace')
        endif
        clmdrv%clm2open=1
      endif

      write(60,996)' Statistical Summary of CLM Output for: ', &
        ld%t%mo,'/',ld%t%da,'/',ld%t%yr,ld%t%hr,:', &
        ld%t%mn,:',ld%t%ss
996   format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
      write(60,*)
      write(60,997)
997   format(t26,'Mean',t40,'StDev',t54,'Min',t68,'Max')

      if(ld%o%wtile.eq.1) then
        call clm2_tileout(ld,tile,57,60)
      else
        call clm2_gridout(ld,tile,57,60)
      endif
    endif
  endif

```

1.34 Fortran: Module Interface clm2drv_module.F90 (Source File: clm2drv_module.F90)

Module for runtime specific CLM2 variables

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module clm2drv_module
```

ARGUMENTS:

```
type clmdrvdec
    integer :: numoutc2      !Counts number of output times for CLM2
    integer :: clm2open       ! Keeps track of opening files
    character*40 :: clm2_rfile !CLM2 Active restart file
    character*40 :: clm2_vfile !CLM2 Vegetation Tile Specification File
    character*40 :: clm2_chtfile !CLM2 Canopy Heights File
    real :: clm2_ism          !CLM2 intial soil moisture
    real :: clm2_it           !CLM2 initial soil temperature
    real :: clm2_iscv          !CLM2 Initial snow mass
    real :: writeintc2         !CLM2 Output Interval (hours)
end type clmdrvdec
```

1.34.1 *clm2_dynsetup.F90* (Source File: *clm2_dynsetup.F90*)

Updates the time dependent CLM variables

REVISION HISTORY:

15 Apr 2002: Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_dynsetup()
```

USES:

```
use lisdrv_module, only: lis,tile
use spmdMod, only : masterproc, npes
use clm2pardef_module
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if(masterproc) then
#endif
  call clm2lairead(lis,tile)
#endif
  call MPI_BCAST(lis%p%laiflag,1,MPI_INTEGER,0, &
                 MPI_COMM_WORLD,ierr)
  if(npes > 1 .and. lis%p%laiflag==1) then
    call clm2_scatter
  endif
```

1.34.2 clm2_gather (Source File: *clm2_gather.F90*)

Gathers CLM tiles

REVISION HISTORY:

30 Jan 2003: Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_gather()
```

USES:

```
use clm_varder
use tile_spmdMod
use clm2pardef_module
```

CONTENTS:

```
#if (defined SPMD)
call MPI_GATHERV(clm(1:di_array(iam)),di_array(iam), &
MPI_CLM_STRUCT,clm,di_array,displs,MPI_CLM_STRUCT, &
0,MPI_COMM_WORLD, ierr)
#endif
```

1.34.3 clm2_gridout.F90 (Source File: *clm2_gridout.F90*)

LIS CLM2 data writer: Write CLM2 output in grid space

REVISION HISTORY:

02 Dec 2003: Sujay Kumar; Initial Version

INTERFACE:

```
subroutine clm2_gridout(ld,tile,ftn,ftn_stats)
```

USES:

```
use lis_module
use tile_module
use clm_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, only : nlevsoi
use clm_varmap, only : patchvec
use clm_varder
```

CONTENTS:

```
soilmtc=0.0
delsoilmoist = 0.0
delswe = 0.0
soilmr=0.0
soilwtc=0.0

clm%totfsa=clm%totfsa/float(clm%count)
call tile2grid(clm%totfsa,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Net Surface Longwave Radiation (W/m2)
!-----
clm%toteflx_lwrad_net=-1.0*clm%toteflx_lwrad_net/float(clm%count)
call tile2grid(clm%toteflx_lwrad_net,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Latent Heat Flux (W/m2)
!-----
clm%toteflx_lh_tot=clm%toteflx_lh_tot/float(clm%count)
call tile2grid(clm%toteflx_lh_tot,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Sensible Heat Flux (W/m2)
!-----
clm%toteflx_sh_tot=clm%toteflx_sh_tot/float(clm%count)
call tile2grid(clm%toteflx_sh_tot,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Ground Heat Flux (W/m2)
!-----
clm%toteflx_soil_grnd=clm%toteflx_soil_grnd/float(clm%count)
call tile2grid(clm%toteflx_soil_grnd,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! General Water Balance Components (Time Averaged)
! Snowfall (kg/m2s)
!-----
clm%totssnow = clm%totssnow/float(clm%count)
call tile2grid(clm%totssnow,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Rainfall (kg/m2s)
!-----
clm%totrain = clm%totain/float(clm%count)
call tile2grid(clm%totain,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp
!-----
! Total Evaporation (kg/m2s)
```

```

!-----
  clm%totqflx_evap = clm%totqflx_evap/float(clm%count)
  call tile2grid(clm%totqflx_evap,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp
!-----

! Surface Runoff (kg/m2s)
!-----
  clm%totqflx_surf = clm%totqflx_surf/float(clm%count)
  call tile2grid(clm%totqflx_surf,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp
!-----

! Subsurface Runoff (kg/m2s)
!-----
  clm%totqflx_drain = clm%totqflx_drain/float(clm%count)
  call tile2grid(clm%totqflx_drain,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp
!-----

! Snowmelt (kg/m2s)
!-----
  snowmelt=clm%totqflx_snowmelt/float(clm%count)
  call tile2grid(snowmelt,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp
!-----

! Calculation of total column soil moisture
! Total soil moisture (liquid+ice) in each layer
!-----
  do m=1,nlevsoi
    do t=1,ld%d%glbnch
      soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
    enddo
  enddo

  do m=1,nlevsoi
    do t=1,ld%d%glbnch
      soilmtc(t)=soilmtc(t)+soilm(t,m)
    enddo
  enddo
!-----

! Change in Soil Moisture (kg/m2)
!-----
  delsoilm=soilmtc-clm%soilmtc_prev
  call tile2grid(delsoilm,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp
!-----

! Change in Snow water equivalent (kg/m2)
!-----
  delswe = clm%h2osno-clm%h2osno_prev
  call tile2grid(delswe,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)

```

```

    write(ftn) gtmp
!-----
! Surface State Variables
! Average Surface Temperature Calculation
!-----
do t=1,ld%d%glbnch
  snowt(t)=0.
  if (clm(t)%itypwat/=istwet)then
    if(clm(t)%snl < 0)then
      snowt(t)=clm(t)%t_soisno(clm(t)%snl+1)
    endif
  endif
  if(snowt(t)==0.)snowt(t)=ld%d%udef
enddo
!-----
! AvgSurfT is the average surface temperature which depends on
! the snow temperature, bare soil temperature and canopy temperature
!-----
do t=1,ld%d%glbnch
  if(snowt(t).ne.ld%d%udef)then
    asurft(t)=clm(t)%frac_sno*snowt(t)+ &
      clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
      (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
      clm(t)%t_grnd
  else
    asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
      (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
  endif
enddo

clm%totqflx_ecanop=clm%totqflx_ecanop/float(clm%count)
cantrn=(clm%totqflx_tran_veg/float(clm%count))
bare=(clm%totqflx_evap_grnd/float(clm%count))
snowevp=(clm%totqflx_sub_snow/float(clm%count))
potevp=ld%d%udef
!-----
! Snow Temperature Calculation
!-----
do t=1,ld%d%glbnch
  snowtemp(t)=0.
  if (clm(t)%itypwat/=istwet)then
    if(clm(t)%snl < 0)then
      totaldepth(t)=0.
      do i=clm(t)%snl+1,0      ! Compute total depth of snow layers
        totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
      enddo
      do i=clm(t)%snl+1,0      ! Compute snow temperature

```

```

        snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
    enddo
    snowtemp(t)=snowtemp(t)/totaldepth(t)
    endif
    if(snowtemp(t).eq.0)snowtemp(t)=ld%d%udef
    endif
enddo
!-----
! Snow Temperature (K)
!-----
call tile2grid(snowtemp,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Canopy Temperature(K)
!-----
call tile2grid(clm%t_veg,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Bare Soil Surface Temperature(K)
!-----
call tile2grid(clm%t_grnd,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Average Surface Temperature(K)
!-----
call tile2grid(asurft,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Effective Radiative Surface Temperature (K)
!-----
call tile2grid(clm%t_rad,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Surface Albedo
!-----
call tile2grid(clm%surfalb,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Snow Water Equivalent (kg/m2)
!-----
call tile2grid(clm%h2osno,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Subsurface State Variables
! Average Layer Soil Moisture (kg/m2)
!-----
do m=1,nlevsoi
    do c=1,ld%d%glbnch

```

```

        tempvar(c)=soilm(c,m)
    enddo
    call tile2grid(tempvar,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
    write(ftn) gtmp
enddo
!-----
! Total Soil Wetness
! Calculation of Total column soil wetness and root zone soil wetness
! soilwtc = (vertically averaged soilm - wilting point)/
!           (vertically averaged layer porosity - wilting point)
! where average soilm is swetint, the wilting point is swetwilt,
! and avgwatsat is average porosity.
! totaldepth represents the total depth of all of the layers
!-----
do t=1,ld%d%glbnch
    swetint(t)=0.
    swetintr(t)=0.
    totaldepth(t)=0.
    avgwatsat(t)=0.
    do m=1,nlevsoi
        avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
        totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
        swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
        swetintr(t)=swetintr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
    enddo
    avgwatsat(t)=avgwatsat(t)/totaldepth(t)
    swetint(t)=(swetint(t)/denh2o)/totaldepth(t)
    swetintr(t)=(swetintr(t)/denh2o)/totaldepth(t)
    soilwtc(t)=100.*swetint(t)/avgwatsat(t)
enddo

call tile2grid(soilwtc,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Evaporation Components
! Vegetation Transpiration (kg/m2s)
!-----
call tile2grid(cantrn,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Bare Soil Evaporation (kg/m2s)
!-----
call tile2grid(bare,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Root Zone Soil Moisture (kg/m2)
! Calculation of root zone soil moisture
!-----

```

```
do t=1,ld%d%glnch
  soilmr(t)=0.
  do m=1,nlevsoi
    soilmr(t)=soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
  enddo
enddo

call tile2grid(soilmr,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Aerodynamic Conductance (m/s)
!-----
call tile2grid(clm%acond,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
if(ld%o%wfor .eq. 1) then
!-----
! Wind (m/s)
!-----
call tile2grid(sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
  gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Rainf (kg/m2s)
!-----
call tile2grid(clm%forc_rain,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Snowf (kg/m2s)
!-----
call tile2grid(clm%forc_snow,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Tair (K)
!-----
call tile2grid(clm%forc_t,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! Qair (kg/kg)
!-----
call tile2grid(clm%forc_q,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! PSurf (Pa)
!-----
call tile2grid(clm%forc_pbot,gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp
!-----
! SWdown (W/m2)
```

```

!-----
    call tile2grid(clm%forc_solad(1)*100.0/35.0,gtmp, &
      ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
    write(ftn) gtmp
!-----
! LWdown (W/m2)
!-----
    call tile2grid(clm%forc_lwrad,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
    write(ftn) gtmp
end if
close(ftn)

call stats(clm%totfsa,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
  vmax)
write(ftn_stats,999)'Swnet (W/m2): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%toteflx_lwrad_net,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Lwnet (W/m2): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%toteflx_lh_tot,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qle (W/m2): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%toteflx_sh_tot,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qh (W/m2): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%toteflx_soil_grnd,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qg (W/m2): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%totsnow,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
  vmax)
write(ftn_stats,998)'Snowf (kg/m2s): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%totrain,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
  vmax)
write(ftn_stats,998)'Rainf (kg/m2s): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%totqflx_evap,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Evap (kg/m2s): ', &
  vmean,vstdev,vmin,vmax
call stats(clm%totqflx_surf,ld%d%udef,ld%d%glbnch, &
  vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qs (kg/m2s): ', &
  vmean,vstdev,vmin,vmax

```

```

call stats(clm%totqflx_drain,ld%d%undef,ld%d%glbnch, &
           vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qsb (kg/m2s): ', &
           vmean,vstdev,vmin,vmax
call stats(clm%totqflx_snomelt/2.5e6,ld%d%undef,ld%d%glbnch, &
           vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qsm (kg/m2s): ', &
           vmean,vstdev,vmin,vmax
call stats(delsoilmoist,ld%d%undef,ld%d%glbnch, &
           vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelSoilMoist (kg/m2): ', &
           vmean,vstdev,vmin,vmax
call stats(delswe,ld%d%undef,ld%d%glbnch, &
           vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelSWE (kg/m2): ', &
           vmean,vstdev,vmin,vmax
call stats(snowtemp,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SnowT (K): ', &
           vmean,vstdev,vmin,vmax
call stats(clm%t_veg,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,999)'VegT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_grnd,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,999)'BaresoilT (K): ',vmean,vstdev,vmin,vmax
call stats(asurft,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'AvgSurfT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_rad,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,999)'RadT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%surfalb,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,999)'Albedo (-): ',vmean,vstdev,vmin,vmax
call stats(clm%h2ocan,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,998)'SWE (kg/m2):      ,vmean,vstdev,vmin,vmax

do m=1,nlevsoi
  do c=1,ld%d%glbnch
    tempvar(c)=soilm(c,m)
  enddo
  call stats(tempvar,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,995)'SoilMoist',m,' (kg/m2): ',vmean,vstdev,vmin,vmax
enddo

call stats(soilwtc,ld%d%undef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SoilWet (%): ',vmean,vstdev,vmin,vmax
call stats(cantrn/2.5e6,ld%d%undef,ld%d%glbnch, &

```

```

      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'TVeg (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
call stats(bare,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'ESoil (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
call stats(soilmr,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'RootMoist (kg/m2): ', &
      vmean,vstdev,vmin,vmax
call stats(clm%acond,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,998)'ACond (m/s): ', &
      vmean,vstdev,vmin,vmax
if(ld%o%wfor.eq.1) then
  call stats(sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
    ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'Wind(m/s): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_rain,ld%d%udef,ld%d%glbnch, &
    vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'Rainf(kg/m2s): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_snow,ld%d%udef,ld%d%glbnch, &
    vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'Snowf(kg/m2s): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_t,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
    vmax)
  write(ftn_stats,999)'Tair(K): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_q,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
    vmax)
  write(ftn_stats,999)'Qair(kg/kg): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_pbot,ld%d%udef,ld%d%glbnch,vmean, &
    vstdev,vmin,vmax)
  write(ftn_stats,999)'PSurf(Pa): ',&
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_solad(1)*100.0/35.0,ld%d%udef,ld%d%glbnch, &
    vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'SWdown(W/m2): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%forc_lwrad,ld%d%udef,ld%d%glbnch,vmean, &
    vstdev,vmin,vmax)
  write(ftn_stats,999)'LWdown(W/m2): ', &
      vmean,vstdev,vmin,vmax

```

```

        endif
995 format (1x,a10,I1,a9,4f14.3)
999 format (1x,a15,4f14.3)
998 format (1x,a15,4e14.3)

```

1.34.4 clmlairead.F90: (Source File: clm2lairead.F90)

This program reads in AVHRR LAI data for CLM

REVISION HISTORY:

```

27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data

```

INTERFACE:

```
subroutine clm2lairead (ld,tile)
```

USES:

```

use clm_varder
use lis_module          ! LIS parameters
use tile_module         ! LIS Tile variables
use clmtype             ! 1-D CLM variables
use shr_const_mod, only : SHR_CONST_PI
use time_manager
use lisdrv_module, only : grid

```

CONTENTS:

```

!-----
! Determine current time to find correct LAI files
!-----
if (ld%t%tscount .eq. 0) then
  ld%t%yr = ld%t%syr
  ld%t%mo = ld%t%smo
  ld%t%da = ld%t%sda
  ld%t%mn = ld%t%smn
  ld%t%ss = ld%t%sss
else
  ld%t%yr = ld%t%yr
  ld%t%mo = ld%t%mo
  ld%t%da = ld%t%da
  ld%t%mn = ld%t%mn
  ld%t%ss = ld%t%ss
endif

```

```

call date2time(ld%t%time,ld%t%doy,ld%t%gmt,ld%t%yr, &
    ld%t%mo,ld%t%da,ld%t%hr,ld%t%mn,ld%t%ss)
!-----
! Initialize LAI flag variable
!-----
ld%p%laiflag = 0

zeroi=0
numi=16
!-----
! Determine Monthly data Times (Assume Monthly
! value valid at DA=16 HR=00Z)
!-----
if (ld%t%da .lt. 16) then
    mo1 = ld%t%mo-1
    yr1 = ld%t%yr
    if (mo1 .eq. 0) then
        mo1 = 12
        yr1 = ld%t%yr - 1
    endif
    mo2 = ld%t%mo
    yr2 = ld%t%yr
else
    mo1 = ld%t%mo
    yr1 = ld%t%yr
    mo2 = ld%t%mo+1
    yr2 = ld%t%yr
    if (mo2 .eq. 13) then
        mo2 = 1
        yr2 = ld%t%yr + 1
    endif
endif
endif

call date2time(time1,doy1,gmt1,yr1,mo1,numi,zeroi,zeroi,zeroi)
call date2time(time2,doy2,gmt2,yr2,mo2,numi,zeroi,zeroi,zeroi)
!-----
! Check to see if need new LAI data
!-----
if (time2 .gt. ld%p%laitime) then
    ld%p%laiflag = 1
else
    ld%p%laiflag = 0
endif

avhrrdir = ld%p%avhrrdir
!-----
! Get new LAI data if required

```

```

!-----
if (ld%p%laiflag .eq. 1) then
print*, 'in clmlai read.'
write(unit=temp,fmt='(i4,i2.2)') yr1, mo1
read (unit=temp,fmt='(a4,a2)') cyr1, cmo1
write(unit=temp,fmt='(i4,i2.2)') yr2, mo2
read (unit=temp,fmt='(a4,a2)') cyr2, cmo2

ld%p%laitime = time2
if(ld%d%domain ==6) then
  select case (ld%p%lai)
  case(1)
    call ncarlai_clm2(name9,name10,name11,name12,name13,name14,name15,name16, &
                      ld%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
  case(2)
    call avhrr_file_5km(name,name2,name3,name4,name5,name6,name7,name8, &
                         name9,name10,name11,name12,name13,name14,name15,name16, &
                         ld%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
  case(3)
    call modis_file_2(name9,name10,name11,name12,name13,name14,name15,name16, &
                      ld%p%modisdir,cyr1,cyr2,cmo1,cmo2)
  case default
    print*, "not a valid lai option"
    call endrun
  end select
else
  select case (ld%p%lai)
  case(1)
    call ncarlai_clm2(name9,name10,name11,name12,name13,name14,name15,name16, &
                      ld%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
  case(2)
    call avhrr_file_2(name,name2,name3,name4,name5,name6,name7,name8, &
                      name9,name10,name11,name12,name13,name14,name15,name16, &
                      ld%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
  case(3)
    call modis_file_2(name9,name10,name11,name12,name13,name14,name15,name16, &
                      ld%p%modisdir,cyr1,cyr2,cmo1,cmo2)
  case default
    print*, "not a valid lai option"
    call endrun
  end select
endif
!-----
! Open AVHRR LAI files (assumes realtime monthly files are present first
! then uses climatology files)
! Assume realtime monthly files are present as default
!-----

```

```

flag1 = 0
flag2 = 0
if(ld%d%domain ==6) then
    open(10,file=name9,status='old',form='unformatted',&
         access='direct',recl=22,iostat=ios1)
    open(11,file=name10,status='old',form='unformatted',&
         access='direct',recl=22,iostat=ios2)
    open(12,file=name13,status='old',form='unformatted',&
         access='direct',recl=22,iostat=ios1)
    open(13,file=name14,status='old',form='unformatted',&
         access='direct',recl=22,iostat=ios2)
else
    open(10,file=name9,status='old',form='unformatted',&
         access='direct',recl=24,iostat=ios1)
    open(11,file=name10,status='old',form='unformatted',&
         access='direct',recl=24,iostat=ios2)
    open(12,file=name13,status='old',form='unformatted',&
         access='direct',recl=24,iostat=ios1)
    open(13,file=name14,status='old',form='unformatted',&
         access='direct',recl=24,iostat=ios2)
endif
print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 1 ', &
        name9,' (',iam,')'
print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 2 ', &
        name10,' (',iam,')'

if (ios1 .ne. 0) then
    close(10)
    if(ld%d%domain==6) then
        print*, 'open 10 .',name11
        open(10,file=name11,status='old',form='unformatted',&
              access='direct',recl=22)
        close(12)
        open(12,file=name15,status='old',form='unformatted',&
              access='direct',recl=22)
    print*, 'msg: clm2lairead -- no realtime monthly data for month 1',&
          ' (',iam,')'
    print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 1 ', &
          name11,' (',iam,')'
    flag1 = 1
else
    print*, 'open 10 2 ',name11
    open(10,file=name11,status='old',form='unformatted',&
          access='direct',recl=24)
    close(12)
    open(12,file=name15,status='old',form='unformatted',&
          access='direct',recl=24)
print*, 'msg: clm2lairead -- no realtime monthly data for month 1',&

```

```

        ' (',iam,')
print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 1 ', &
       name11, ' (',iam,')
flag1 = 1
endif
endif

if (ios2 .ne. 0) then
  close(11)
  if( ld%d%domain ==6) then
    open(11,file=name12,status='old',form='unformatted',&
         access='direct',recl=22)
    close(13)
    open(13,file=name16,status='old',form='unformatted',&
         access='direct',recl=22)
    flag2 = 1
    print*, 'msg: clm2lairead -- no realtime monthly data for month 2', &
           ' (',iam,')
    print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 2 ', &
           name12, ' (',iam,')
  else
    open(11,file=name12,status='old',form='unformatted',&
         access='direct',recl=24)
    close(13)
    open(13,file=name16,status='old',form='unformatted',&
         access='direct',recl=24)
    flag2 = 1
    print*, 'msg: clm2lairead -- no realtime monthly data for month 2', &
           ' (',iam,')
    print*, 'msg: clm2lairead -- using 1/8 avhrr lai/dsai data for month 2 ', &
           name12, ' (',iam,')
  endif
endif

do t=1,ld%d%nch
  latdeg = clm(t)%lat*180/shr_const_pi
  londeg = clm(t)%lon*180/shr_const_pi
  if (ld%d%domain .ne. 1 .and. ld%d%domain .le. 5) then

    select case (ld%d%domain)
    case (2)
      d_start_nr  = ((latdeg - (-59.875)) / 0.25) + 1
      start_8th_nr = ((d_start_nr - 1) * 2) + 1
      end_8th_nr   =   start_8th_nr + 1
      d_start_nc   = ((londeg - (-179.875)) / 0.25) + 1
      start_8th_nc = ((d_start_nc - 1) * 2) + 1
      end_8th_nc   =   start_8th_nc + 1
    case (3)

```

```

d_start_nr  = ((latdeg - (-60)) / 2.0) + 1
start_8th_nr = ((d_start_nr - 1) * 16) + 1
end_8th_nr   = start_8th_nr + 15
d_start_nc   = ((londeg - (-180)) / 2.5) + 1
start_8th_nc = ((d_start_nc - 1) * 20) + 1
end_8th_nc   = start_8th_nc + 19
case (4)
  d_start_nr  = ((latdeg - (-59.500)) / 1.00) + 1
  start_8th_nr = ((d_start_nr - 1) * 8) + 1
  end_8th_nr   = start_8th_nr + 7
  d_start_nc   = ((londeg - (-179.500)) / 1.00) + 1
  start_8th_nc = ((d_start_nc - 1) * 8) + 1
  end_8th_nc   = start_8th_nc + 7
case (5)
  d_start_nr  = ((latdeg - (-59.750)) / 0.50) + 1
  start_8th_nr = ((d_start_nr - 1) * 4) + 1
  end_8th_nr   = start_8th_nr + 3
  d_start_nc   = ((londeg - (-179.750)) / 0.50) + 1
  start_8th_nc = ((d_start_nc - 1) * 4) + 1
  end_8th_nc   = start_8th_nc + 3
case default
  print*, 'err: clm2lairead -- improper domain selection', (',iam,')
  call endrun
end select
!-----
! Initialize sums for LAI month 1, LAI month 2, DSAI month 1, DSAI month 2
!-----
sum1 = 0.0
sum2 = 0.0
sum3 = 0.0
sum4 = 0.0
cnt1 = 0
cnt2 = 0
cnt3 = 0
cnt4 = 0
!-----
! Looping over 1/8 grid space that relates to 1/4 or 2x2.5 domains
!-----
do ii8 = start_8th_nr,end_8th_nr
  do j8 = start_8th_nc,end_8th_nc
    line = (ii8 - 1)*2880 + j8
    read(10,rec=line) lat1, lon1, lai_1
    read(12,rec=line) lat1, lon1, sai_1
    read(11,rec=line) lat2, lon2, lai_2
    read(13,rec=line) lat2, lon2, sai_2
    select case (ld%p%lai)

```

```

        case(2)      ! avhrr lai

        if (ichar(lai_1(clm(t)%itypveg+1)) .ne. 251 &
            .and. ichar(lai_1(clm(t)%itypveg+1)) .ne. 0) then
            sum1 = sum1 + (ichar(lai_1(clm(t)%itypveg+1))) * 0.04
            cnt1 = cnt1 + 1
        endif
        if (ichar(sai_1(clm(t)%itypveg+1)) .ne. 251 &
            .and. ichar(sai_1(clm(t)%itypveg+1)) .ne. 0) then
            sum3 = sum3 + (ichar(sai_1(clm(t)%itypveg+1))) * 0.04
            cnt3 = cnt3 + 1
        endif
        if (ichar(lai_2(clm(t)%itypveg+1)) .ne. 251 &
            .and. ichar(lai_2(clm(t)%itypveg+1)) .ne. 0) then
            sum2 = sum2 + (ichar(lai_2(clm(t)%itypveg+1))) * 0.04
            cnt2 = cnt2 + 1
        endif
        if (ichar(sai_2(clm(t)%itypveg+1)) .ne. 251 &
            .and. ichar(sai_2(clm(t)%itypveg+1)) .ne. 0) then
            sum4 = sum4 + (ichar(sai_2(clm(t)%itypveg+1))) * 0.04
            cnt4 = cnt4 + 1
        endif

        case(3)      ! modis lai
        if (ichar(lai_1(clm(t)%itypveg+1)) .lt. 200) then
            sum1 = sum1 + (ichar(lai_1(clm(t)%itypveg+1))) * 0.10
            cnt1 = cnt1 + 1
        endif
        if (ichar(sai_1(clm(t)%itypveg+1)) .lt. 200) then
            sum3 = sum3 + (ichar(sai_1(clm(t)%itypveg+1))) * 0.10
            cnt3 = cnt3 + 1
        endif
        if (ichar(lai_2(clm(t)%itypveg+1)) .lt. 200) then
            sum2 = sum2 + (ichar(lai_2(clm(t)%itypveg+1))) * 0.10
            cnt2 = cnt2 + 1
        endif
        if (ichar(sai_2(clm(t)%itypveg+1)) .lt. 200) then
            sum4 = sum4 + (ichar(sai_2(clm(t)%itypveg+1))) * 0.10
            cnt4 = cnt4 + 1
        endif
        case default
            print*, 'err: clm2lairead -- not a valid lai domain', ',iam,'
            call endrun

        end select
    enddo
enddo
!-----

```

```

! Compute averages for the vegetation type represented by tile
!-----
      if (cnt1 .ne. 0) then
        lai_t1_f(t) = sum1 / cnt1
      else
        lai_t1_f(t) = 0
      endif
      if (cnt2 .ne. 0) then
        lai_t2_f(t) = sum2 / cnt2
      else
        lai_t2_f(t) = 0
      endif
      if (cnt3 .ne. 0) then
        sai_t1_f(t) = sum3 / cnt3
      else
        sai_t1_f(t) = 0
      endif
      if (cnt4 .ne. 0) then
        sai_t2_f(t) = sum4 / cnt4
      else
        sai_t2_f(t) = 0
      endif
    else
      ! MLAT = (LATDEG - (-59.9375)) / 0.125 + 1
      ! MLON = (LONDEG - (-179.9375)) / 0.125 + 1
      ! LINE = (MLAT - 1)*2880 + MLON
      mlat = (latdeg - (-59.975+0.05/2)) / 0.05 + 1
      mlon = (londeg - (-179.975+0.05/2)) / 0.05 + 1
      line = (mlat - 1)*7200 + mlon
      read(10,rec=line) lat1, lon1, lai_1
      read(12,rec=line) lat1, lon1, sai_1
      read(11,rec=line) lat2, lon2, lai_2
      read(13,rec=line) lat2, lon2, sai_2
      select case(1d%p%lai)
      case(2) ! avhrr lai
        lai_t1_f(t) = ichar(lai_1(clm(t)%itypveg+1)) * 0.04
        sai_t1_f(t) = ichar(sai_1(clm(t)%itypveg+1)) * 0.04
        lai_t2_f(t) = ichar(lai_2(clm(t)%itypveg+1)) * 0.04
        sai_t2_f(t) = ichar(sai_2(clm(t)%itypveg+1)) * 0.04
      case(3)
        lai_t1_f(t) = ichar(lai_1(clm(t)%itypveg+1)) * 0.10
        sai_t1_f(t) = ichar(sai_1(clm(t)%itypveg+1)) * 0.10
        lai_t2_f(t) = ichar(lai_2(clm(t)%itypveg+1)) * 0.10
        sai_t2_f(t) = ichar(sai_2(clm(t)%itypveg+1)) * 0.10
      case default
        print*, 'err: clm2lairead -- invalid domain for lai data', ',iam,'
        call endrun
      end select

```

```

        endif
    enddo
    close(10)
    close(11)
    close(12)
    close(13)

!-----
! Determine weights between months
!-----
wt1= (time2-ld%t%time)/(time2-time1)
wt2= (ld%t%time-time1)/(time2-time1)
!-----
! Assign interpolated LAI and DSAI values to the CLM variable names
! used in CLM main
!-----
do t=1,ld%d%nch
    clm(t)%tlai = wt1 * lai_t1_f(t) + wt2 * lai_t2_f(t)
    clm(t)%tsai = wt1 * sai_t1_f(t) + wt2 * sai_t2_f(t)
    if (clm(t)%itypveg .eq. 12) then
        clm(t)%tlai=0.0
        clm(t)%tsai=0.0
        clm(t)%htop=0.0
        clm(t)%hbot=0.0
    endif
    print*, t,clm(t)%tlai,clm(t)%tsai
enddo

if(ld%o%wparam.eq.1) then
    allocate(domlai(ld%d%lnc,ld%d%lnr))
    domlai = 0.0
    do t=1,ld%d%nch
        if(grid(t)%lat*1000.ge.ld%d%kgds(4).and. &
           grid(t)%lat*1000.le.ld%d%kgds(7).and. &
           grid(t)%lon*1000.ge.ld%d%kgds(5).and. &
           grid(t)%lon*1000.le.ld%d%kgds(8)) then
            rindex = tile(t)%row - (ld%d%kgds(4)-ld%d%kgds(44)) &
            /ld%d%kgds(9)
            cindex = tile(t)%col - (ld%d%kgds(5)-ld%d%kgds(45)) &
            /ld%d%kgds(10)
            domlai(cindex,rindex) = clm(t)%tlai
        endif
    enddo

    open(32,file="domlai.bin",form='unformatted')
    write(32) domlai
    close(32)
    deallocate(domlai)
endif

```

```
endif
end subroutine clm2lairead
```

1.34.5 avhrr_file_2 (Source File: clm2lairead.F90)

This subroutine puts together AVHRR file name

INTERFACE:

```
subroutine avhrr_file_2 (NAME,NAME2,NAME3,NAME4,NAME5,NAME6,NAME7,NAME8, &
    NAME9,NAME10,NAME11,NAME12,NAME13,NAME14,NAME15,NAME16, &
    avhrrdir, cyr1, cyr2, cmo1, cmo2 )
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a9)
67 format (a15)
98 format (a1, a4, a2)
66 format (a5,a2)
99 format (7a1)

write(unit=temp1,fmt='(a40)') avhrrdir
write(unit=temp2,fmt='(a40)') avhrrdir
write(unit=temp3,fmt='(a40)') avhrrdir
write(unit=temp4,fmt='(a40)') avhrrdir

read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)

write(unit=temp1,fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2,fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4,fmt='(7a1)') fdir_4

do i = 1, 7
```

```

if ( fdir(i) == ' ' ) fdir(i) = '0'
if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp2, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp3, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp4, fmt='(a15)') '_AVHRRLAI_0.125'
read (unit=temp1, fmt='(80a1)') (fsubsn(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_2(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_3(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_4(i), i=1,15)
write(unit=temp1, fmt='(a15)') '_AVHRRSAI_0.125'
write(unit=temp2, fmt='(a15)') '_AVHRRSAI_0.125'
write(unit=temp3, fmt='(a15)') '_AVHRRSAI_0.125'
write(unit=temp4, fmt='(a15)') '_AVHRRSAI_0.125'
read (unit=temp1, fmt='(80a1)') (fsubsn_5(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_6(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_7(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
    if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
    if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
    if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
    if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
    (fsubsn(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
    (fsubsn_2(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
    (fsubsn_3(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
    (fsubsn_4(i), i=1,15)
read(unit=temp1, fmt='(a80)') name9
read(unit=temp2, fmt='(a80)') name10
read(unit=temp3, fmt='(a80)') name11
read(unit=temp4, fmt='(a80)') name12
write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
    (fsubsn_5(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
    (fsubsn_6(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
    (fsubsn_7(i), i=1,15)

```

```

        (fsubsn_7(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
        (fsubsn_8(i), i=1,15)
read(unit=temp1, fmt='(a80)') name13
read(unit=temp2, fmt='(a80)') name14
read(unit=temp3, fmt='(a80)') name15
read(unit=temp4, fmt='(a80)') name16

return

```

1.34.6 avhrr_file_5KM (Source File: clm2lairead.F90)

This subroutine puts together AVHRR file name

INTERFACE:

```

subroutine avhrr_file_5KM (NAME,NAME2,NAME3,NAME4,NAME5,NAME6,NAME7,NAME8,  &
    NAME9,NAME10,NAME11,NAME12,NAME13,NAME14,NAME15,NAME16,  &
    avhrrdir, cyr1, cyr2, cmo1, cmo2 )

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a9)
67 format (a8)
68 format (a12)
98 format (a1, a4, a2)
66 format (a5,a2)
99 format (7a1)

        write(unit=temp1, fmt='(a40)') avhrrdir
        write(unit=temp2, fmt='(a40)') avhrrdir
        write(unit=temp3, fmt='(a40)') avhrrdir
        write(unit=temp4, fmt='(a40)') avhrrdir
        read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
        read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
        read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
        read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)

        write(unit=temp1, fmt='(a1,a4,a2)') ' / ', cyr1, cmo1
        read(unit=temp1, fmt='(7a1)') fdir
        write(unit=temp2, fmt='(a1,a4,a2)') ' / ', cyr2, cmo2
        read(unit=temp2, fmt='(7a1)') fdir_2

```

```

write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4, fmt='(7a1)') fdir_4

do i = 1, 7
  if ( fdir(i) == ' ') fdir(i) = '0'
  if ( fdir_2(i) == ' ') fdir_2(i) = '0'
  if ( fdir_3(i) == ' ') fdir_3(i) = '0'
  if ( fdir_4(i) == ' ') fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a8)') '_5KM.bin'
write(unit=temp2, fmt='(a8)') '_5KM.bin'
write(unit=temp3, fmt='(a8)') '_5KM.bin'
write(unit=temp4, fmt='(a8)') '_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_2(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_3(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_4(i), i=1,15)
write(unit=temp1, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp2, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp3, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp4, fmt='(a12)') '_SAI_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn_5(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_6(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_7(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
  (fsubsn(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
  (fsubsn_2(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
  (fsubsn_3(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
  (fsubsn_4(i), i=1,15)
read(unit=temp1, fmt='(a80)') name9
read(unit=temp2, fmt='(a80)') name10
read(unit=temp3, fmt='(a80)') name11

```

```

read(unit=temp4, fmt='(a80)') name12
write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
  (fsubsn_5(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
  (fsubsn_6(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
  (fsubsn_7(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
  (fsubsn_8(i), i=1,15)
read(unit=temp1, fmt='(a80)') name13
read(unit=temp2, fmt='(a80)') name14
read(unit=temp3, fmt='(a80)') name15
read(unit=temp4, fmt='(a80)') name16
print*, 'n11',name11
print*, 'n15',name15
return

```

1.34.7 ncarlai_clm2 (Source File: *clm2lairead.F90*)

This subroutine puts together NCAR LAI/SAI/TOP/BOT file names

INTERFACE:

```

subroutine ncarlai_clm2 (NAME9,NAME10,NAME11,NAME12,NAME13,NAME14,NAME15,NAME16,  &
    avhrrdir, cyr1, cyr2, cmo1, cmo2 )

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a9)
67 format (a15)
98 format (a1, a4, a2)
66 format (a5,a2)
99 format (7a1)

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
open(unit=91, file='temp_2', form='formatted', access='direct', recl=80)
open(unit=92, file='temp_3', form='formatted', access='direct', recl=80)
open(unit=93, file='temp_4', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) avhrrdir
write(91, 96, rec=1) avhrrdir
write(92, 96, rec=1) avhrrdir
write(93, 96, rec=1) avhrrdir

```

```

read(90, 92, rec=1) (fbase(i), i=1,80)
read(91, 92, rec=1) (fbase_2(i), i=1,80)
read(92, 92, rec=1) (fbase_3(i), i=1,80)
read(93, 92, rec=1) (fbase_4(i), i=1,80)

write(90, 98, rec=1) ' ', cyr1, cmo1
read(90, 99, rec=1) fdir
write(91, 98, rec=1) ' ', cyr2, cmo2
read(91, 99, rec=1) fdir_2
write(92, 66, rec=1) '/CLIM', cmo1
read(92, 99, rec=1) fdir_3
write(93, 66, rec=1) '/CLIM', cmo2
read(93, 99, rec=1) fdir_4

do i = 1, 7
  if ( fdir(i) == ' ') fdir(i) = '0'
  if ( fdir_2(i) == ' ') fdir_2(i) = '0'
  if ( fdir_3(i) == ' ') fdir_3(i) = '0'
  if ( fdir_4(i) == ' ') fdir_4(i) = '0'
enddo

write(90, 67, rec=1) '_NCLM2LAI_0.125'
write(91, 67, rec=1) '_NCLM2LAI_0.125'
write(92, 67, rec=1) '_NCLM2LAI_0.125'
write(93, 67, rec=1) '_NCLM2LAI_0.125'
read (90, 92, rec=1) (fsubsn(i), i=1,15)
read (91, 92, rec=1) (fsubsn_2(i), i=1,15)
read (92, 92, rec=1) (fsubsn_3(i), i=1,15)
read (93, 92, rec=1) (fsubsn_4(i), i=1,15)
write(90, 67, rec=1) '_NCLM2SAI_0.125'
write(91, 67, rec=1) '_NCLM2SAI_0.125'
write(92, 67, rec=1) '_NCLM2SAI_0.125'
write(93, 67, rec=1) '_NCLM2SAI_0.125'
read (90, 92, rec=1) (fsubsn_5(i), i=1,15)
read (91, 92, rec=1) (fsubsn_6(i), i=1,15)
read (92, 92, rec=1) (fsubsn_7(i), i=1,15)
read (93, 92, rec=1) (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7), &
                     (fsubsn(i), i=1,15)

```

```

write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
                      (fsubsn_2(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
                      (fsubsn_3(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
                      (fsubsn_4(i), i=1,15)
read(90, 93, rec=1) name9
read(91, 93, rec=1) name10
read(92, 93, rec=1) name11
read(93, 93, rec=1) name12
write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
                      (fsubsn_5(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
                      (fsubsn_6(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
                      (fsubsn_7(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
                      (fsubsn_8(i), i=1,15)
read(90, 93, rec=1) name13
read(91, 93, rec=1) name14
read(92, 93, rec=1) name15
read(93, 93, rec=1) name16

close(90)
close(91)
close(92)
close(93)
return

```

1.34.8 ncacanht_clm2 (Source File: *clm2lairead.F90*)

This subroutine puts together NCAR TOP/BOT file names

INTERFACE:

```

subroutine ncacanht_clm2 (NTOP1,NTOP2,NBOT1,NBOT2,  &
                          avhrrdir, cyr1, cyr2, cmo1, cmo2 )

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a9)
67 format (a15)

```

```

98 format (a1, a4, a2)
66 format (a5,a2)
99 format (7a1)

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
open(unit=91, file='temp_2', form='formatted', access='direct', recl=80)
open(unit=92, file='temp_3', form='formatted', access='direct', recl=80)
open(unit=93, file='temp_4', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) avhrrdir
write(91, 96, rec=1) avhrrdir
write(92, 96, rec=1) avhrrdir
write(93, 96, rec=1) avhrrdir
read(90, 92, rec=1) (fbase(i), i=1,80)
read(91, 92, rec=1) (fbase_2(i), i=1,80)
read(92, 92, rec=1) (fbase_3(i), i=1,80)
read(93, 92, rec=1) (fbase_4(i), i=1,80)

write(90, 66, rec=1) '/CLIM', cmo1
read(90, 99, rec=1) fdir
write(91, 66, rec=1) '/CLIM', cmo2
read(91, 99, rec=1) fdir_2
write(92, 66, rec=1) '/CLIM', cmo1
read(92, 99, rec=1) fdir_3
write(93, 66, rec=1) '/CLIM', cmo2
read(93, 99, rec=1) fdir_4

do i = 1, 7
  if ( fdir(i) == ' ') fdir(i) = '0'
  if ( fdir_2(i) == ' ') fdir_2(i) = '0'
  if ( fdir_3(i) == ' ') fdir_3(i) = '0'
  if ( fdir_4(i) == ' ') fdir_4(i) = '0'
enddo

write(90, 67, rec=1) '_NCLM2TOP_0.125'
write(91, 67, rec=1) '_NCLM2TOP_0.125'
write(92, 67, rec=1) '_NCLM2BOT_0.125'
write(93, 67, rec=1) '_NCLM2BOT_0.125'
read (90, 92, rec=1) (fsubsn(i), i=1,15)
read (91, 92, rec=1) (fsubsn_2(i), i=1,15)
read (92, 92, rec=1) (fsubsn_3(i), i=1,15)
read (93, 92, rec=1) (fsubsn_4(i), i=1,15)
c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

```

```

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
                     (fsubsn(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
                     (fsubsn_2(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
                     (fsubsn_3(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
                     (fsubsn_4(i), i=1,15)
read(90, 93, rec=1) ntop1
read(91, 93, rec=1) ntop2
read(92, 93, rec=1) nbot1
read(93, 93, rec=1) nbot2

close(90)
close(91)
close(92)
close(93)

return

```

1.34.9 modis_file_2 (Source File: clm2lairead.F90)

This subroutine puts together MODIS file name

INTERFACE:

```

subroutine modis_file_2 (NAME9,NAME10,NAME11,NAME12,NAME13,NAME14,NAME15,NAME16,  &
    modisdir, cyr1, cyr2, cmo1, cmo2 )

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a9)
67 format (a15)
98 format (a1, a4, a2)
66 format (a5,a2)
99 format (7a1)

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
open(unit=91, file='temp_2', form='formatted', access='direct', recl=80)
open(unit=92, file='temp_3', form='formatted', access='direct', recl=80)
open(unit=93, file='temp_4', form='formatted', access='direct', recl=80)

```

```

write(90, 96, rec=1) modisdir
write(91, 96, rec=1) modisdir
write(92, 96, rec=1) modisdir
write(93, 96, rec=1) modisdir
read(90, 92, rec=1) (fbase(i), i=1,80)
read(91, 92, rec=1) (fbase_2(i), i=1,80)
read(92, 92, rec=1) (fbase_3(i), i=1,80)
read(93, 92, rec=1) (fbase_4(i), i=1,80)

write(90, 98, rec=1) '/', cyr1, cmo1
read(90, 99, rec=1) fdir
write(91, 98, rec=1) '/', cyr2, cmo2
read(91, 99, rec=1) fdir_2
write(92, 66, rec=1) '/CLIM', cmo1
read(92, 99, rec=1) fdir_3
write(93, 66, rec=1) '/CLIM', cmo2
read(93, 99, rec=1) fdir_4

do i = 1, 7
  if ( fdir(i) == ' ') fdir(i) = '0'
  if ( fdir_2(i) == ' ') fdir_2(i) = '0'
  if ( fdir_3(i) == ' ') fdir_3(i) = '0'
  if ( fdir_4(i) == ' ') fdir_4(i) = '0'
enddo

write(90, 67, rec=1) '_MODISLAI_0.125'
write(91, 67, rec=1) '_MODISLAI_0.125'
write(92, 67, rec=1) '_MODISLAI_0.125'
write(93, 67, rec=1) '_MODISLAI_0.125'
read (90, 92, rec=1) (fsubsn(i), i=1,15)
read (91, 92, rec=1) (fsubsn_2(i), i=1,15)
read (92, 92, rec=1) (fsubsn_3(i), i=1,15)
read (93, 92, rec=1) (fsubsn_4(i), i=1,15)
write(90, 67, rec=1) '_MODISSAI_0.125'
write(91, 67, rec=1) '_MODISSAI_0.125'
write(92, 67, rec=1) '_MODISSAI_0.125'
write(93, 67, rec=1) '_MODISSAI_0.125'
read (90, 92, rec=1) (fsubsn_5(i), i=1,15)
read (91, 92, rec=1) (fsubsn_6(i), i=1,15)
read (92, 92, rec=1) (fsubsn_7(i), i=1,15)
read (93, 92, rec=1) (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
  if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1

```

```

end do

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
                      (fsubsn(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
                      (fsubsn_2(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
                      (fsubsn_3(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
                      (fsubsn_4(i), i=1,15)
read(90, 93, rec=1) name9
read(91, 93, rec=1) name10
read(92, 93, rec=1) name11
read(93, 93, rec=1) name12

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
                      (fsubsn_5(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
                      (fsubsn_6(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
                      (fsubsn_7(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
                      (fsubsn_8(i), i=1,15)
read(90, 93, rec=1) name13
read(91, 93, rec=1) name14
read(92, 93, rec=1) name15
read(93, 93, rec=1) name16

close(90)
close(91)
close(92)
close(93)

return

```

1.34.10 *clm2_output.F90* (Source File: *clm2_output.F90*)

This subroutines sets up methods to write CLM2 output

INTERFACE:

```
subroutine clm2_output
```

USES:

```
use lisdrv_module, only : lis, tile
use clm_varctl, only : clmdrv
use spmdMod, only : masterproc, npes
```

CONTENTS:

```

if(lis%wsingle==1) then
!-----
! Write output with each variable in a single file
!-----
if(mod(lis%gmt,clmdrv%writeintc2).eq.0)then
    do i=1,11
        call clm2_singlegather(i,var)
        if(masterproc) then
            call clm2_singleout(lis, tile, var, i)
        endif
    enddo
    do i=14,35
        call clm2_singlegather(i, var)
        if(masterproc) then
            call clm2_singleout(lis, tile, var,i)
        endif
    enddo
    call clm2_totinit()
endif
else
!-----
! Write bundled output
!-----
if(mod(lis%gmt,clmdrv%writeintc2).eq.0)then
    if(npes > 1 ) then
        call clm2_gather()
    endif
    if(masterproc) then
        call clm2_almaout(lis, tile)
    endif
    call clm2_totinit()
endif
endif

```

1.35 Fortran: Module Interface *clm2pardef_module.F90* (Source File: *clm2pardef_module.F90*)

This module contains routines that defines MPI derived data types for CLM LSM

REVISION HISTORY:

06 Oct 2003; Sujay Kumar Initial Specification

INTERFACE:

```
module clm2pardef_module
```

USES:

```
use clmtype
use clm2drv_module
use spmdMod
```

1.35.1 def_clmpar_struct (Source File: *clm2pardef_module.F90*)

Routine that defines MPI derived data types for CLM

INTERFACE:

```
subroutine def_clmpar_struct()
```

1.35.2 *clm2_restart.F90* (Source File: *clm2_restart.F90*)

This program reads the restart files for CLM

REVISION HISTORY:

20 Jan 2003; Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_restart(rw)
```

USES:

```
use spmdMod, only : masterproc,npes
use lisdrv_module, only : lis
use restFileMod , only : restrd
```

CONTENTS:

```
if(masterproc) then
  if(rw.eq.1 .and. lis%o%startcode.eq.1) then
    print*, 'Reading restart files..'
    call restrd()
  endif
endif
```

1.35.3 clm2_scatter.F90 (Source File: clm2_scatter.F90)

Distributes clm2 tiles on to compute nodes

REVISION HISTORY:

Apr 2003 ; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine clm2_scatter()
```

USES:

```
use clm_varder
use tile_spmdMod
use clm2pardef_module
```

1.35.4 clm2_setup.F90 (Source File: clm2_setup.F90)

Completes the CLM2 setup routines.

REVISION HISTORY:

20 Jan 2003 Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_setup()
```

USES:

```
use lisdrv_module, only: lis, tile
use spmdMod
use time_manager
use clm_varder
use clm_varcon, only: eccen, obliqr, lambm0 , mvelpp
use clm_varctl, only: nsrest
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endif
```

```
! -----
! Get current date
! -----
```

```
if ( masterproc ) then
  call get_curr_date(yr, mon, day, ncsec)
```

```

        endif
! -----
! If initial run: initialize time-varying data
! If continuation run: end of initialization because time varying
! read in from restart file
! -----

      if (nsrest == 0) then
        call canhtset()
        if (masterproc) then
          write (6,*) ('Attempting to initialize time variant variables .....')
        endif

        readini = .false.
        call iniTimeVar (readini, eccen, obliqr, lambm0 , mvelpp, lis, tile)

        if (masterproc) then
          write (6,*) ('Successfully initialized time variant variables')
          write (6,*)
        endif

      endif

! -----
! End initialization
! -----
      if (masterproc) then
        write (6,*) ('Successfully initialized the land model')
        if (nsrest == 0) then
          write (6,*) 'begin initial run at: '
        else
          write (6,*) 'begin continuation run at:'
        end if
        write (6,*), nstep= ,get_nstep(), &
          ' year= ',yr,' month= ',mon,' day= ',day,' seconds= ',ncsec
        write (6,*)
        write (6, '(72a1)') ("*",i=1,60)
        write (6,*)
      endif
      clm%totfsa=0.           ! solar absorbed solar radiation [W/m2]
      clm%toteflx_lwrad_net=0. ! net longwave radiation [W/m2]
      clm%toteflx_lh_tot=0.    ! total latent heat flux [W/m2]
      clm%toteflx_sh_tot=0.    ! total sensible heat flux [W/m2]
      clm%toteflx_soil_grnd=0. ! ground heat flux [W/m2]
      clm%totqflx_snomelt=0.   ! snowmelt heat flux [W/m2]
      clm%totrain=0.           ! accumulation of rain [mm]
      clm%totsnow=0.            ! accumulation of snow [mm]

```

```

clm%totqflx_evap=0.          ! total evaporation [mm]
clm%totqflx_surf=0.          ! surface runoff [mm]
clm%totqflx_drain=0.         ! subsurface runoff [mm]
clm%totqflx_ecanop=0.        ! interception evaporation [W/m2]
clm%totqflx_tran_veg=0.
clm%totqflx_evap_grnd=0.
clm%totqflx_sub_snow=0.
clm%count=0
clm%acond=0.
#endif ( ! defined OPENDAP )
#endif
if ( npes > 1 ) then
    call clm2_scatter
endif
#endif
return

```

1.35.5 clm2_singlegather.F90 (Source File: clm2_singlegather.F90)

Gather single variable for output

REVISION HISTORY:

Apr 2003 ; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine clm2_singlegather(index, var)
```

USES:

```

use lisdrv_module, only : lis
use clm_varcon, ONLY : istwet,denh2o
use clm_varpar, ONLY : nlevsoi
use clm_varder
use tile_spmdMod
use clm2pardef_module

```

CONTENTS:

```

if(index ==1) then
    var_temp = clm%totfsa/float(clm%count)
elseif (index ==2) then
    var_temp = -1.0*clm%toteflx_lwrad_net/float(clm%count)
elseif(index == 3) then
    var_temp = clm%toteflx_lh_tot/float(clm%count)
elseif(index ==4) then
    var_temp = clm%toteflx_sh_tot/float(clm%count)

```

```

elseif(index ==5) then
  var_temp = clm%toteflx_soil_grnd/float(clm%count)
elseif(index ==6) then
  var_temp = clm%totsnow/float(clm%count)
elseif(index ==7) then
  var_temp = clm%totrain/float(clm%count)
elseif(index ==8) then
  var_temp = clm%totqflx_evap/float(clm%count)
elseif(index ==9) then
  var_temp = clm%totqflx_surf/float(clm%count)
elseif (index ==10) then
  var_temp = clm%totqflx_drain/float(clm%count)
elseif (index ==11) then
  var_temp = clm%totqflx_snomelt/float(clm%count)
elseif (index ==14) then !SnowT
  do t=1,di_array(iam)
    snowtemp(t)=0.
    if (clm(t)%itypwat/=istwet)then
      if(clm(t)%snl < 0)then
        totaldepth(t)=0.
        do i=clm(t)%snl+1,0      ! Compute total depth of snow layers
          totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
        enddo
        do i=clm(t)%snl+1,0      ! Compute snow temperature
          snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
        enddo
        snowtemp(t)=snowtemp(t)/totaldepth(t)
      endif
      if(snowtemp(t).eq.0)snowtemp(t)=lis%d%udef
    endif
  enddo
  var_temp = snowtemp
elseif (index ==15) then !VegT
  var_temp = clm%t_veg
elseif(index ==16) then !BareSoilT
  var_temp = clm%t_grnd
elseif (index ==17) then !AvgSurfT
  do t=1,di_array(iam)
    snowt(t) = 0.0
    if(clm(t)%itypwat /=istwet) then
      if(clm(t)%snl <0) then
        snowt(t) = clm(t)%t_soisno(clm(t)%snl+1)
      endif
    endif
    if(snowt(t)==0.0) snowt(t) = lis%d%udef

    if(snowt(t).ne.lis%d%udef)then
      asurft(t)=clm(t)%frac_sno*snowt(t)+ &

```

```

        clm(t)%frac_veg_nosno*clm(t)%t_veg+  &
        (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
        clm(t)%t_grnd
    else
        asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
        (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
    endif
enddo
var_temp = asurft
elseif (index ==18) then !AvgSurfT
    var_temp = clm%t_rad
elseif(index ==19) then !Albedo
    var_temp = clm%surfalb
elseif(index ==20) then !SWE
    var_temp = clm%h2osno
elseif(index ==21) then !SoilMoist1
    var_temp = clm%h2osoii_liq(1)+clm%h2osoii_ice(1)
elseif(index ==22) then !SoilMoist2
    var_temp = clm%h2osoii_liq(2)+clm%h2osoii_ice(2)
elseif(index ==23) then !SoilMoist3
    var_temp = clm%h2osoii_liq(3)+clm%h2osoii_ice(3)
elseif(index ==24) then !SoilMoist4
    var_temp = clm%h2osoii_liq(4)+clm%h2osoii_ice(4)
elseif(index ==25) then !SoilMoist5
    var_temp = clm%h2osoii_liq(5)+clm%h2osoii_ice(5)
elseif(index ==26) then !SoilMoist6
    var_temp = clm%h2osoii_liq(6)+clm%h2osoii_ice(6)
elseif(index ==27) then !SoilMoist7
    var_temp = clm%h2osoii_liq(7)+clm%h2osoii_ice(7)
elseif(index ==28) then !SoilMoist8
    var_temp = clm%h2osoii_liq(8)+clm%h2osoii_ice(8)
elseif(index ==29) then !SoilMoist9
    var_temp = clm%h2osoii_liq(9)+clm%h2osoii_ice(9)
elseif(index ==30) then !SoilMoist10
    var_temp = clm%h2osoii_liq(10)+clm%h2osoii_ice(10)
elseif(index==31) then !Soilwet
    do t=1,di_array(iam)
        swetint(t) = 0.0
        avgwatsat(t) = 0.0
        totaldepth(t) = 0.0
        do m=1,nlevsoi
            avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
            totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
            swetint(t)=swetint(t)+clm(t)%h2osoii_liq(m)
        enddo
        avgwatsat(t) = avgwatsat(t)/totaldepth(t)
        swetint(t) = (swetint(t)/denh2o)/totaldepth(t)
        var_temp(t) = 100*swetint(t)/avgwatsat(t)
    enddo

```

```

        enddo
elseif(index==32) then !TVeg
    var_temp = clm%totqflx_tran_veg/float(clm%count)
elseif(index==33) then !ESoil
    var_temp = clm%totqflx_evap_grnd/float(clm%count)
elseif(index==34) then !RootMoist
    do t=1,di_array(iam)
        soilmr(t) = 0.0
        do m=1,nlevsoi
            soilmr(t) = soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
        enddo
    enddo
    var_temp = soilmr
elseif(index==35) then !ACond
    var_temp = clm%acond
endif

call MPI_GATHERV(var_temp(1:di_array(iam)),di_array(iam), &
    MPI_REAL,var,di_array,displs,MPI_REAL, &
    0,MPI_COMM_WORLD, ierr)

```

1.35.6 clm2_singleout.F90 (Source File: clm2_singleout.F90)

Write output file for a single CLM variable

REVISION HISTORY:

14 Jun 2002; Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_singleout (ld,tile, var_array, index)
```

USES:

```

use lis_module      ! LDAS non-model-specific 1-D variables
use tile_module
use clm_varcon, ONLY : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, ONLY : nlevsoi
use clm_varmap, ONLY : patchvec
use clm_varctl, only : clmdrv

```

CONTENTS:

```
!-----
! Test to see if output writing interval has been reached
!-----
if(mod(ld%t%gmt,clmdrv%writeintc2).eq.0)then
```

```

!-----
! Generate directory structure and file names for CLM Output
!-----

      length = len(trim(vname1(index)))
      WRITE(UNIT=temp, FMT='(A10)') VNAME1(index)
      READ(UNIT=temp,FMT='(10A1)') (FVARNAME(I), I=1,length)
      WRITE(unit=temp,fmt='(I4,I2,I2)')LD%T%YR,LD%T%MO,LD%T%DA
      READ(unit=temp,fmt='(8a1)')FTIME
      DO I=1,8
         IF(FTIME(I).EQ.(' '))FTIME(I)='0'
      ENDDO

      WRITE(unit=temp,fmt='(I4)')LD%T%YR
      READ(unit=temp,fmt='(8a1)')FTIMEC
      DO I=1,4
         IF(FTIMEC(I).EQ.(' '))FTIMEC(I)='0'
      ENDDO

      WRITE(unit=temp,fmt='(a7,i3,a1)')'/LDAS.E',LD%0%EXPCODE,'.
      READ(unit=temp,fmt='(80a1)') (FNAME(I),I=1,11)
      DO I=1,11
         IF(FNAME(I).EQ.(' '))FNAME(I)='0'
      ENDDO

      WRITE(unit=temp,fmt='(a40)') LD%0%ODIR
      READ(unit=temp,fmt='(40a1)') (FBASE(I),I=1,40)
      C=0
      DO I=1,40
         IF(FBASE(I).EQ.(' ').AND.C.EQ.0)C=I-1
      ENDDO

      WRITE(unit=temp,fmt='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
         LD%0%EXPCODE,'/CLM2/ ', &
         LD%t%YR,'/ ',LD%T%YR,LD%T%MO,LD%T%DA
      READ(unit=temp,fmt='(80A1)') (FYRMODIR(I),I=1,26)
      DO I=1,26
         IF(FYRMODIR(I).EQ.(' '))FYRMODIR(I)='0'
      ENDDO

      WRITE(unit=temp,fmt='(A9)')'mkdir -p '
      READ(unit=temp,fmt='(80A1)')(FMKDIR(I),I=1,9)

      WRITE(unit=temp,fmt='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
         (FYRMODIR(I),I=1,26)
      READ(unit=temp,fmt='(A80)')MKFYRMO

!-----
! Make the directories for the CLM2 output files

```

```

!-----
      call system(mkfyrmo)
!-----
! Generate file name for binary output
!-----
      if(ld%o%wout.eq.1)then
        write(unit=temp,fmt='(I4,I2,I2,I2)')ld%t%yr, &
          ld%t%mo,ld%t%da,ld%t%hr
        read(unit=temp,fmt='(10A1)')ftimeb
        do i=1,10
          if(ftimeb(i).eq.(' '))ftimeb(i)='0'
        enddo

        write(unit=temp,fmt='(A9)')'.CLM2gbin'
        read(unit=temp,fmt='(80A1)') (fsubgb(i),i=1,9)

        write(unit=temp,fmt='(80A1)')(fbase(i),i=1,c), &
          (FYRMODIR(I),I=1,26), &
          (fname(i),i=1,11),(ftimeb(i),i=1,10), &
          (fvarname(i),i=1,length),(fsubgb(i),i=1,9 )
        read(unit=temp,fmt='(A80)')filengb
      endif
      if(ld%o%wout.eq.1)then
        open(57,file=filengb,form='unformatted')

        clmdrv%numoutc2=clmdrv%numoutc2+1
        call t2gr(var_array,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
        write(57) gtmp
!-----
! Write statistical output
!-----
      if(clmdrv%clm2open.eq.0)then
        file='CLMstats.dat'
        call openfile(name,ld%o%odir,ld%o%expcode,file)
        if(ld%o%startcode.eq.1)then
          open(60,file=name,form='formatted',status='unknown', &
            position='append')
        else
          open(60,file=name,form='formatted',status='replace')
        endif
        clmdrv%clm2open=1
      endif

      write(60,996)'      Statistical Summary of CLM Output for: ', &
        ld%t%mo,'/',ld%t%da,'/',ld%t%yr,ld%t%hr,:', &
        ld%t%mn,:',ld%t%ss

996      format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)

```

```

997      format(t26,'Mean',t40,'StDev',t54,'Min',t68,'Max')

      call stats(var_array,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
      write(60,999) vname(index),vmean,vstdev,vmin,vmax
      endif
      endif
      endif
995 format (1x,a10,I1,a9,4f14.3)
999 format (1x,a15,4f14.3)
998 format (1x,a15,4e14.3)

```

1.35.7 clm2_tileout.F90 (Source File: clm2_tileout.F90)

LIS CLM2 data writer: Write CLM2 output in tile space

REVISION HISTORY:

02 Dec 2003: Sujay Kumar; Initial Version

INTERFACE:

```
subroutine clm2_tileout(ld,tile,ftn,ftn_stats)
```

USES:

```

use lis_module
use tile_module
use clm_varcon, ONLY : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, ONLY : nlevsoi
use clm_varmap, ONLY : patchvec
use clm_varder

```

CONTENTS:

```

soilmtc=0.0
delsoilmoist = 0.0
delswe = 0.0
soilmr=0.0
soilwtc=0.0
!-----
! Net Surface Shortwave Radiation (W/m2)
!-----
      clm%totfsa=clm%totfsa/float(clm%count)
      call t2gr(clm%totfsa,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
      write(ftn) gtmp
!-----
! Net Surface Longwave Radiation (W/m2)

```

```

!-----
  clm%toteflx_lwrad_net=-1.0*clm%toteflx_lwrad_net/float(clm%count)
  call t2gr(clm%toteflx_lwrad_net,gtmp,ld%d%glbngrid,ld%d%glbnch, &
             tile)
  write(ftn) gtmp
!-----

! Latent Heat Flux (W/m2)
!-----
  clm%toteflx_lh_tot=clm%toteflx_lh_tot/float(clm%count)
  call t2gr(clm%toteflx_lh_tot,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp
!-----

! Sensible Heat Flux (W/m2)
!-----
  clm%toteflx_sh_tot=clm%toteflx_sh_tot/float(clm%count)
  call t2gr(clm%toteflx_sh_tot,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp
!-----

! Ground Heat Flux (W/m2)
!-----
  clm%toteflx_soil_grnd=clm%toteflx_soil_grnd/float(clm%count)
  call t2gr(clm%toteflx_soil_grnd,gtmp,ld%d%glbngrid,ld%d%glbnch, &
             tile)
  write(ftn) gtmp
!-----

! General Water Balance Components (Time Averaged)
! Snowfall (kg/m2s)
!-----
  clm%totsnow = clm%totsnow/float(clm%count)
  call t2gr(clm%totsnow,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp
!-----

! Rainfall (kg/m2s)
!-----
  clm%totrain = clm%totrain/float(clm%count)
  call t2gr(clm%totrain,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp
!-----

! Total Evaporation (kg/m2s)
!-----
  clm%totqflx_evap = clm%totqflx_evap/float(clm%count)
  call t2gr(clm%totqflx_evap,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp
!-----

! Surface Runoff (kg/m2s)
!-----
  clm%totqflx_surf = clm%totqflx_surf/float(clm%count)
  call t2gr(clm%totqflx_surf,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)

```

```

    write(ftn) gtmp
!-----
! Subsurface Runoff (kg/m2s)
!-----
    clm%totqflx_drain = clm%totqflx_drain/float(clm%count)
    call t2gr(clm%totqflx_drain,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Snowmelt (kg/m2s)
!-----
    snowmelt=clm%totqflx_snomelt/float(clm%count)
    call t2gr(snowmelt,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Calculation of total column soil moisture
! Total soil moisture (liquid+ice) in each layer
!-----
    do m=1,nlevsoi
        do t=1,ld%d%glbnch
            soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
        enddo
    enddo

    do m=1,nlevsoi
        do t=1,ld%d%glbnch
            soilmtc(t)=soilmtc(t)+soilm(t,m)
        enddo
    enddo
!-----
! Change in Soil Moisture (kg/m2)
!-----
    delsoilmoist = soilmtc-clm%soilmtc_prev
    call t2gr(delsoilmoist,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Change in Snow water equivalent (kg/m2)
!-----
    delswe = clm%h2osno-clm%h2osno_prev
    call t2gr(delswe,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Surface State Variables
! Average Surface Temperature Calculation
!-----
    do t=1,ld%d%glbnch
!-----
! SnowT is the snow surface temperature, i.e. top layer t_soisno
!-----

```

```

snowt(t)=0.
if (clm(t)%itypwat/=istwet)then
    if(clm(t)%snl < 0)then
        snowt(t)=clm(t)%t_soisno(clm(t)%snl+1)
    endif
endif
if(snowt(t)==0.)snowt(t)=ld%d%udef
enddo
!-----
! AvgSurfT is the average surface temperature which depends on
! the snow temperature, bare soil temperature and canopy temperature
!-----
do t=1,ld%d%glnch
    if(snowt(t).ne.ld%d%udef)then
        asurft(t)=clm(t)%frac_sno*snowt(t)+ &
            clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
            (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
            clm(t)%t_grnd
    else
        asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
            (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
    endif
enddo

clm%totqflx_ecanop=clm%totqflx_ecanop/float(clm%count)
cantrn=(clm%totqflx_tran_veg/float(clm%count))
bare=(clm%totqflx_evap_grnd/float(clm%count))
snowevp=(clm%totqflx_sub_snow/float(clm%count))
potevp=ld%d%udef
!-----
! Snow Temperature Calculation
!-----
do t=1,ld%d%glnch
    snowtemp(t)=0.
    if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
            totaldepth(t)=0.
            do i=clm(t)%snl+1,0      ! Compute total depth of snow layers
                totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
            enddo
            do i=clm(t)%snl+1,0      ! Compute snow temperature
                snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
            enddo
            snowtemp(t)=snowtemp(t)/totaldepth(t)
        endif
        if(snowtemp(t).eq.0)snowtemp(t)=ld%d%udef
    endif
enddo

```

```
    enddo
!-----
! Snow Temperature (K)
!-----
    call t2gr(snowtemp,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Canopy Temperature(K)
!-----
    call t2gr(clm%t_veg,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Bare Soil Surface Temperature(K)
!-----
    call t2gr(clm%t_grnd,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Average Surface Temperature(K)
!-----
    call t2gr(asurft,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Effective Radiative Surface Temperature (K)
!-----
    call t2gr(clm%t_rad,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Surface Albedo
!-----
    call t2gr(clm%surfalb,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Snow Water Equivalent (kg/m2)
!-----
    call t2gr(clm%h2osno,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
    write(ftn) gtmp
!-----
! Subsurface State Variables
! Average Layer Soil Moisture (kg/m2)
!-----
    do m=1,nlevsoi
        do c=1,ld%d%glbnch
            tempvar(c)=soilm(c,m)
        enddo
        call t2gr(tempvar,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
        write(ftn) gtmp
    enddo
!-----
```

```

! Total Soil Wetness
! Calculation of Total column soil wetness and root zone soil wetness
! soilwtc = (vertically averaged soilm - wilting point)/
!           (vertically averaged layer porosity - wilting point)
! where average soilm is swetint, the wilting point is swetwilt,
! and avgwatsat is average porosity.
! totaldepth represents the total depth of all of the layers
!-----
do t=1,ld%d%glbnch
  swetint(t)=0.
  swetintr(t)=0.
  totaldepth(t)=0.
  avgwatsat(t)=0.
  do m=1,nlevsoi
    avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
    totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
    swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
    swetintr(t)=swetintr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
  enddo
  avgwatsat(t)=avgwatsat(t)/totaldepth(t)
  swetint(t)=(swetint(t)/denh2o)/totaldepth(t)
  swetintr(t)=(swetintr(t)/denh2o)/totaldepth(t)
  soilwtc(t)=100.*swetint(t)/avgwatsat(t)
enddo

call t2gr(soilwtc,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Evaporation Components
! Vegetation Transpiration (kg/m2s)
!-----
call t2gr(cantrn,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Bare Soil Evaporation (kg/m2s)
!-----
call t2gr(bare,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Root Zone Soil Moisture (kg/m2)
! Calculation of root zone soil moisture
!-----
do t=1,ld%d%glbnch
  soilmr(t)=0.
  do m=1,nlevsoi
    soilmr(t)=soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
  enddo
enddo

```

```
call t2gr(soilmr,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Aerodynamic Conductance (m/s)
!-----
call t2gr(clm%acond,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
if(ld%o%wfor .eq. 1) then
!-----
! Wind (m/s)
!-----
call t2gr(sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Rainf (kg/m2s)
!-----
call t2gr(clm%forc_rain,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Snowf (kg/m2s)
!-----
call t2gr(clm%forc_snow,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Tair (K)
!-----
call t2gr(clm%forc_t,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! Qair (kg/kg)
!-----
call t2gr(clm%forc_q,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! PSurf (Pa)
!-----
call t2gr(clm%forc_pbot,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! SWdown (W/m2)
!-----
call t2gr(clm%forc_solad(1)*100.0/35.0,gtmp, &
ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp
!-----
! LWdown (W/m2)
```

```

!-----
      call t2gr(clm%forc_lwrad,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
      write(ftn) gtmp
      endif
      close(ftn)
      call stats(clm%totfsa,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
                 vmax)
      write(ftn_stats,999)'SWnet (W/m2): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%toteflx_lwrad_net,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,999)'LWnet (W/m2): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%toteflx_lh_tot,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,999)'Qle (W/m2): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%toteflx_sh_tot,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,999)'Qh (W/m2): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%toteflx_soil_grnd,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,999)'Qg (W/m2): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totsnow,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
                 vmax)
      write(ftn_stats,998)'Snowf (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totrain,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
                 vmax)
      write(ftn_stats,998)'Rainf (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totqflx_evap,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,998)'Evap (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totqflx_surf,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,998)'Qs (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totqflx_drain,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,998)'Qsb (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
      call stats(clm%totqflx_snomelt/2.5e6,ld%d%udef,ld%d%glbnch, &
                 vmean,vstdev,vmin,vmax)
      write(ftn_stats,998)'Qsm (kg/m2s): ', &

```

```

      vmean,vstdev,vmin,vmax
call stats(delsoilmoist,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelSoilMoist (kg/m2): ', &
      vmean,vstdev,vmin,vmax
call stats(delswe,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelsWE (kg/m2): ', &
      vmean,vstdev,vmin,vmax
call stats(snowtemp,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SnowT (K): ', &
      vmean,vstdev,vmin,vmax
call stats(clm%t_veg,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,999)'VegT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_grnd,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,999)'BaresoilT (K): ',vmean,vstdev,vmin,vmax
call stats(asurft,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'AvgSurfT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_rad,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,999)'RadT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%surfalb,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,999)'Albedo (-): ',vmean,vstdev,vmin,vmax
call stats(clm%h2ocan,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
      vmax)
write(ftn_stats,998)'SWE (kg/m2):           ,vmean,vstdev,vmin,vmax

do m=1,nlevsoi
  do c=1,ld%d%glbnch
    tempvar(c)=soilm(c,m)
  enddo
  call stats(tempvar,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,995)'SoilMoist',m,' (kg/m2): ',vmean,vstdev,vmin,vmax
enddo

call stats(soilwtc,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SoilWet (%): ',vmean,vstdev,vmin,vmax
call stats(cantrn/2.5e6,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'TVeg (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
call stats(bare,ld%d%udef,ld%d%glbnch, &
      vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'ESoil (kg/m2s): ', &
      vmean,vstdev,vmin,vmax

```

```

call stats(soilmr,ld%d%udef,ld%d%glbnch, &
           vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'RootMoist (kg/m2): ', &
           vmean,vstdev,vmin,vmax
call stats(clm%acond,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
           vmax)
write(ftn_stats,998)'ACond (m/s): ', &
           vmean,vstdev,vmin,vmax
if(ld%o%wfor.eq.1) then
  call stats(sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
             ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'Wind(m/s): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_rain,ld%d%udef,ld%d%glbnch, &
             vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'Rainf(kg/m2s): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_snow,ld%d%udef,ld%d%glbnch, &
             vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'Snowf(kg/m2s): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_t,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
             vmax)
  write(ftn_stats,999)'Tair(K): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_q,ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, &
             vmax)
  write(ftn_stats,999)'Qair(kg/kg): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_pbot,ld%d%udef,ld%d%glbnch,vmean, &
             vstdev,vmin,vmax)
  write(ftn_stats,999)'PSurf(Pa): ',&
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_solad(1)*100.0/35.0,ld%d%udef,ld%d%glbnch, &
             vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'SWdown(W/m2): ', &
           vmean,vstdev,vmin,vmax
  call stats(clm%forc_lwrad,ld%d%udef,ld%d%glbnch,vmean, &
             vstdev,vmin,vmax)
  write(ftn_stats,999)'LWdown(W/m2): ', &
           vmean,vstdev,vmin,vmax
endif
995 format (1x,a10,I1,a9,4f14.3)
999 format (1x,a15,4f14.3)
998 format (1x,a15,4e14.3)

```

1.35.8 clm2_totinit.F90 (Source File: clm2_totinit.F90)

Initialize CLM output arrays

REVISION HISTORY:

14 Jun 2002 Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2_totinit()
```

USES:

```
use clm_varder
use tile_spmdMod
```

CONTENTS:

```
do t = 1, di_array(iam)
  clm(t)%totfsa=0.          ! solar absorbed solar radiation [W/m2]
  clm(t)%totefflx_lwrad_net=0. ! net longwave radiation [W/m2]
  clm(t)%totefflx_lh_tot=0.   ! total latent heat flux [W/m2]
  clm(t)%totefflx_sh_tot=0.   ! total sensible heat flux [W/m2]
  clm(t)%totefflx_soil_grnd=0. ! ground heat flux [W/m2]
  clm(t)%totqflx_snomelt=0.   ! snowmelt heat flux [W/m2]
  clm(t)%totrain=0.           ! accumulation of rain [mm]
  clm(t)%totsnow=0.           ! accumulation of snow [mm]
  clm(t)%totqflx_evap=0.      ! total evaporation [mm]
  clm(t)%totqflx_surf=0.      ! surface runoff [mm]
  clm(t)%totqflx_drain=0.     ! subsurface runoff [mm]
  clm(t)%totqflx_ecanop=0.    ! interception evaporation [W/m2]
  clm(t)%totqflx_tran_veg=0.
  clm(t)%totqflx_evap_grnd=0.
  clm(t)%totqflx_sub_snow=0.
  clm(t)%count=0
enddo
soilmtc = 0.0
do m=1,nlevsoi
  do t=1,di_array(iam)
    soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
  enddo
enddo
do m=1,nlevsoi
  do t=1,di_array(iam)
    soilmtc(t)=soilmtc(t)+soilm(t,m)
  enddo
enddo
do t=1,di_array(iam)
  clm(t)%soilmtc_prev = soilmtc(t)
  clm(t)%h2osno_prev = clm(t)%h2osno
enddo
```

!ROUTINE:clm2wrst.F90

This program writes the restart files for CLM

REVISION HISTORY:

20 Jan 2003; Sujay Kumar Initial Specification

INTERFACE:

```
subroutine clm2wrst()
```

USES:

```
use spmdMod, only : masterproc
use restFileMod, only : restwrt
use clm_varctl, only : clmdrv
use lisdrv_module, only : lis
```

CONTENTS:

```
if(masterproc) then
  if((lis%t%gmt.eq.(24-clmdrv%writeintc2)) &
     .or. lis%t%endtime.eq. 1) then
    print*, 'Writing CLM restart..'
    call restwrt()
  endif
endif
```

1.35.9 clm_varder.F90 (Source File: clm_varder.F90)

Module to initialize clm variables

INTERFACE:

```
module clm_varder
```

USES:

```
use clmtype
use spmdMod

implicit none
!ARGUMENTS
type (clm1d), allocatable :: clm(:)
```

1.35.10 clm_varder_ini (Source File: clm_varder.F90)

Reads in runtime clm parameters, allocates memory for variables

INTERFACE:

```
subroutine clm_varder_ini(nch)
```

USES:

```
use infnan
use tile_spmdMod
use clm_varmap
use clm_varcon
use clm_varctl, only : clmdrv
use shr_orb_mod
use initializeMod, only : initialize
use clm2pardef_module
```

CONTENTS:

```
if(masterproc) then
    call readclm2crd(clmdrv)
endif
call def_clmpar_struct
call MPI_BCAST(clmdrv, 1, MPI_CLMDRV_STRUCT, 0, &
    MPI_COMM_WORLD, ier)
!-----
! allocate memory for clm derived type
!-----
if(masterproc) then
    allocate (clm(1:nch))
    print*, 'MSG: clm_varder_ini -- allocating clm',di_array(iam), &
        ' (',iam,)'
    call clm_varder_init(nch)
else
    begland = 1
    endland = di_array(iam)
    begpatch = 1
    endpatch = di_array(iam)
    print*, 'MSG: clm_varder_ini -- allocating clm',di_array(iam), &
        ' (',iam,)'
    allocate(clm(di_array(iam)))
endif
if(masterproc) then
    call shr_orb_params
endif
call MPI_BCAST(numpatch,1,MPI_INTEGER,0,&
    MPI_COMM_WORLD,ier)
call MPI_BCAST(eccen,1,MPI_REAL,0,&
```

```

        MPI_COMM_WORLD,ier)
call MPI_BCAST(obliqr,1,MPI_REAL,0,&
               MPI_COMM_WORLD,ier)
call MPI_BCAST(lambm0,1,MPI_REAL,0,&
               MPI_COMM_WORLD,ier)
call MPI_BCAST(mvelpp,1,MPI_REAL,0,&
               MPI_COMM_WORLD,ier)
#if ( defined OPENDAP )
    call initialize
#else
    if(masterproc) then
        call initialize
    endif
#endif

```

1.35.11 clm_varder_init (Source File: *clm_varder.F90*)

Initializes clm variables

INTERFACE:

```
subroutine clm_varder_init(nch)
```

USES:

```

use infnan
use tile_spmdMod
use clm_varmap

```

1.35.12 iniTimeConst.F90 (Source File: *iniTimeConst.F90*)

Initialize time invariant clm variables

Method:

Author: Gordon Bonan

USES:

```

use precision
use infnan
use clm_varder
use clm_varpar , only : nlevsoi, nlevlak
use clm_varmap , only : begpatch, endpatch, patchvec, numpatch
use clm_varcon , only : istsoil, istice, istdlak, istslak, istwet, spval
use pft_varcon , only : ncorn, nwheat, roota_par, rootb_par, &
                      z0mr, displar, dleaf, rhol, rhos, taul, taus, xl, &
                      qe25, vcmx25, mp, c3psn

```

```

use clm_varsur    , only : zlak, dzlak, zsoi, dzsoi, zisoi

use clm_varctl   , only : nsrest
use time_manager , only : get_step_size
use shr_const_mod, only : SHR_CONST_PI
use spmdMod      , only : masterproc

#if ( defined OPENDAP )
  use opendap_module
#else
  use grid_module       ! LIS Grid variables
#endif
  use lisdrv_module

```

CONTENTS:

```

! -----
! Initialize local variables for error checking
! -----

```

```

sand(:, :) = inf
clay(:, :) = inf

! -----
! itypveg, isoicol, itypwat, sand, and clay from 2-d surface type
! LIS modification: Read in soil files and equate lat,lon
! LIS grid arrays with clm grid
! Reynolds soil parameterization scheme
! Read in soil data maps to gridded arrays
! -----

```

```

#if ( defined OPENDAP )
  print*, 'MSG: iniTimeConst -- Retrieving SOIL file ',      &
          trim(lis%p%safile), ' (',iam,')'
  call system("opendap_scripts/getsand.pl "//ciam// " //"
              trim(lis%p%safile)//" //"
              cparm_slat//" //cparm_nlat//" //"
              cparm_wlon//" //cparm_elon)
  print*, 'MSG: iniTimeConst -- Retrieving CLAY file ',      &
          trim(lis%p%clfile), ' (',iam,')'
  call system("opendap_scripts/getclay.pl "//ciam// " //"
              trim(lis%p%clfile)//" //"
              cparm_slat//" //cparm_nlat//" //"
              cparm_wlon//" //cparm_elon)
  print*, 'MSG: iniTimeConst -- Retrieving COLOR file ',     &
          trim(lis%p%iscfile), ' (',iam,')'
  call system("opendap_scripts/getcolor.pl "//ciam// " //"
              trim(lis%p%iscfile)//" //"
              cparm_slat//" //cparm_nlat//" //"

```

```

        cparm_wlon//" "//cparm_elon)
#endif
print*, 'MSG: iniTimeConst -- Reading soil, clay and color files'
print*, 'MSG: iniTimeConst -- Opening soil file ',lis%p%safile
open(11, file=lis%p%safile, form='unformatted', status='old')
print*, 'MSG: iniTimeConst -- Opening clay file ',lis%p%clfile
open(12, file=lis%p%clfile, form='unformatted', status='old')
print*, 'MSG: iniTimeConst -- Opening color file ',lis%p%iscfile
open(13, file=lis%p%iscfile, form='unformatted', status='old')
read(11) sand1
read(12) clay1
read(13) color
close(11)
close(12)
close(13)
print*, 'MSG: iniTimeConst -- Read soil, clay and color files'
do k = 1,numpatch
if (tile(k)%fgrd > 0.0) then      !valid subgrid patch
    clm(k)%kpatch = k
    clm(k)%itypveg = tile(k)%vegt
    clm(k)%isoicol = int(color(tile(k)%col,tile(k)%row-tnroffset))
    clm(k)%isoicol = color(tile(k)%col,tile(k)%row)
    if (tile(k)%vegt == npatch_urban) then !urban, from pcturb
        clm(k)%itypwat = istsoil
        print*, 'ERR: iniTimeConst -- not supposed to be in urban block'
    else if (tile(k)%vegt == npatch_lake) then !deep lake, from pctlak
        clm(k)%itypwat = istdlak
    do l = 1,nlevsoi
        sand(l,k) = 0._r8
        clay(l,k) = 0._r8
    end do
    else if (tile(k)%vegt == npatch_wet) then !wetland, from pctwet
        clm(k)%itypwat = istwet
    do l = 1,nlevsoi
        sand(l,k) = 0._r8
        clay(l,k) = 0._r8
    end do
    else if (tile(k)%vegt == npatch_gla) then !glacier, from pctgla
        clm(k)%itypwat = istice
        print*, 'ERR: iniTimeConst -- not supposed to be in glacier block'
    else
        !soil
        clm(k)%itypwat = istsoil
    do l = 1, nlevsoi
        sand(l,k)=sand1(tile(k)%col,tile(k)%row-tnroffset)
        clay(l,k)=clay1(tile(k)%col,tile(k)%row-tnroffset)
    enddo
end if
end if

```

```
    enddo
! -----
! tag lake points
! -----

do k = 1,numpatch
  if (clm(k)%itypwat==istdlak .or. clm(k)%itypwat==istslak) then
    clm(k)%lakpoi = .true.
  else
    clm(k)%lakpoi = .false.
  end if
end do

! -----
! latitudes and longitudes
! -----

pi = SHR_CONST_PI
do i=1,nc_working
  do j=1,nr_working
    if(gindex(i,j).ne. -1) then
      k = gindex(i,j)
      clm(k)%lat   = grid(tile(k)%index-grid_offset)%lat * pi/180.
      clm(k)%lon   = grid(tile(k)%index-grid_offset)%lon * pi/180.
    endif
  end do
enddo

! -----
! Define layer structure for soil and lakes
! Vertical profile of snow is initialized in routine iniTimeVar
! -----

if (nlevlak /= nlevsoi) then
  write(6,*)'number of soil levels and number of lake levels must be the same'
  write(6,*)'nlevsoi= ',nlevsoi,' nlevlak= ',nlevlak
  call endrun
endif

dzlak(1) = 1.
dzlak(2) = 2.
dzlak(3) = 3.
dzlak(4) = 4.
dzlak(5) = 5.
dzlak(6) = 7.
dzlak(7) = 7.
dzlak(8) = 7.
dzlak(9) = 7.
```

```

dzlak(10)= 7.

zlak(1) = 0.5
zlak(2) = 1.5
zlak(3) = 4.5
zlak(4) = 8.0
zlak(5) = 12.5
zlak(6) = 18.5
zlak(7) = 25.5
zlak(8) = 32.5
zlak(9) = 39.5
zlak(10)= 46.5

do j = 1, nlevsoi
    zsoi(j) = scalez*(exp(0.5*(j-0.5))-1.)      !node depths
enddo

dzsoi(1) = 0.5*(zsoi(1)+zsoi(2))              !thickness b/n two interfaces
do j = 2,nlevsoi-1
    dzsoi(j)= 0.5*(zsoi(j+1)-zsoi(j-1))
enddo
dzsoi(nlevsoi) = zsoi(nlevsoi)-zsoi(nlevsoi-1)

zisoi(0) = 0.
do j = 1, nlevsoi-1
    zisoi(j) = 0.5*(zsoi(j)+zsoi(j+1))          !interface depths
enddo
zisoi(nlevsoi) = zsoi(nlevsoi) + 0.5*dzsoi(nlevsoi)

do k = 1,numpatch
    if (clm(k)%itypwat == istdlak) then           !assume all lakes are deep lakes
        clm(k)%z(1:nlevlak) = zlak(1:nlevlak)
        clm(k)%dz(1:nlevlak) = dzlak(1:nlevlak)
    else if (clm(k)%itypwat == istslak) then       !shallow lake (not used)
        clm(k)%dz(1:nlevlak) = NaN
        clm(k)%z(1:nlevlak) = NaN
    else                                         !soil, ice, wetland
        clm(k)%z(1:nlevsoi) = zsoi(1:nlevsoi)
        clm(k)%dz(1:nlevsoi) = dzsoi(1:nlevsoi)
        clm(k)%zi(0:nlevsoi) = zisoi(0:nlevsoi)
    endif
end do

! -----
! Initialize root fraction (computing from surface, d is depth in meter):
! Y = 1 -1/2 (exp(-ad)+exp(-bd)) under the constraint that
! Y(d = 0.1m) = 1-beta^(10 cm) and Y(d=d_obs)=0.99 with beta & d_obs
! given in Zeng et al. (1998).

```

```

! -----
! do k = begpatch, endpatch
do k = 1,numpatch
  if (.not. clm(k)%lakpoi) then
    ivt = clm(k)%itypveg
    do j = 1, nlevsoi-1
      clm(k)%rootfr(j) = .5*( exp(-roota_par(ivt)*clm(k)%zi(j-1)) &
                               + exp(-rootb_par(ivt)*clm(k)%zi(j-1)) &
                               - exp(-roota_par(ivt)*clm(k)%zi(j )) &
                               - exp(-rootb_par(ivt)*clm(k)%zi(j )) )
    end do
    clm(k)%rootfr(nlevsoi) = .5*( exp(-roota_par(ivt)*clm(k)%zi(nlevsoi-1)) &
                                    + exp(-rootb_par(ivt)*clm(k)%zi(nlevsoi-1)) )

  else
    clm(k)%rootfr(1:nlevsoi) = spval
  end if
end do

! -----
! Initialize soil thermal and hydraulic properties
! -----
do k = 1,numpatch
  if (clm(k)%itypwat == istsoil) then
    !soil
    do j = 1, nlevsoi
      clm(k)%bsw(j)      = 2.91 + 0.159*clay(j,k)*100.0
      clm(k)%watsat(j)   = 0.489 - 0.00126*sand(j,k)*100.0
      xksat              = 0.0070556 *( 10.**(-0.884+0.0153*sand(j,k)*100.0) ) ! mm/s
      clm(k)%hksat(j)   = xksat * exp(-clm(k)%zi(j)/hkdepth)
      clm(k)%sucsat(j)  = 10. * ( 10.**((1.88-0.0131*sand(j,k)*100.0) )
      tkm                = (8.80*sand(j,k)*100.0+2.92*clay(j,k)*100.0)/(sand(j,k)*100.0+clay(j,k)*100.0)
      bd                 = (1.-clm(k)%watsat(j))*2.7e3
      clm(k)%tkmg(j)    = tkm ** (1.- clm(k)%watsat(j))
      clm(k)%tksatu(j)  = clm(k)%tkmg(j)*0.57**clm(k)%watsat(j)
      clm(k)%tkdry(j)   = (0.135*bd + 64.7) / (2.7e3 - 0.947*bd)
      clm(k)%csol(j)    = (2.128*sand(j,k)*100.0+2.385*clay(j,k)*100.0)/ (sand(j,k)*100.0+clay(j,k)*100.0)
    end do
  else
    !ice, lakes, wetlands
    do j = 1, nlevsoi
      clm(k)%bsw(j)      = spval
      clm(k)%watsat(j)   = spval
      clm(k)%hksat(j)   = spval
      clm(k)%sucsat(j)  = spval
      clm(k)%tkmg(j)    = spval
      clm(k)%tksatu(j)  = spval
      clm(k)%tkdry(j)   = spval
      clm(k)%csol(j)    = spval
    end do
  end if
end do

```

```

        end if
    end do

! -----
! Initialize clm derived type components from pft_varcon to avoid
! indirect addressing and be compatible with offline CLM code
! -----

! do k = begpatch, endpatch
do k = 1,numpatch
    ivt = clm(k)%itypveg
    clm(k)%z0mr    = z0mr(ivt)
    clm(k)%displar = displar(ivt)
    clm(k)%dleaf   = dleaf(ivt)
    clm(k)%xl      = xl(ivt)
    do ib = 1,numrad
        clm(k)%rhol(ib) = rhol(ivt,ib)
        clm(k)%rhos(ib) = rhos(ivt,ib)
        clm(k)%taul(ib) = taul(ivt,ib)
        clm(k)%taus(ib) = taus(ivt,ib)
    end do
    clm(k)%qe25    = qe25(ivt)          ! quantum efficiency at 25c (umol co2 / umol photon)
    clm(k)%vcmx25 = vcmx25(ivt)        ! maximum rate of carboxylation at 25c (umol co2/m**2/s)
    clm(k)%mp      = mp(ivt)           ! slope for conductance-to-photosynthesis relationship
    clm(k)%c3psn   = c3psn(ivt)         ! photosynthetic pathway: 0. = c4, 1. = c3
end do

!Initialize other misc derived type components - note that for a
!restart run, dtim is set in routine restrd()

if (nsrest == 0) then
!    do k = begpatch, endpatch
    if ( masterproc ) then
        tmp_dtime = get_step_size()
    endif
#endif ( defined OPENDAP )
    call MPI_BCAST(tmp_dtime,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
#endif
    do k = 1,numpatch
        !clm(k)%dtim = get_step_size()
        clm(k)%dtim = tmp_dtime
    end do
endif

print*, "DBG: iniTimeConst -- dtim", clm(1)%dtim, (',iam,')

if (masterproc) then
    write(6,*)

```

```

write(6,30)
do j = 1,nlevlak
    write(6,40)zlak(j),dzlak(j)
end do
write(6,*)
write(6,35)
do j = 1,nlevsoi
    write(6,45)zsoi(j),dzsoi(j),zisoi(j)
end do
write(6,50)
write(6,*)
endif

30 format(' ',' lake levels ',' lake thickness(m)')
35 format(' ',' soil levels ',' soil thickness(m)', ' soil interfaces(m)')
40 format(' ',2(f7.3,8x))
45 format(' ',3(f7.3,8x))
50 format(' ','Note: top level soil interface is set to 0')

return

```

1.35.13 iniTimeVar.F90 (Source File: iniTimeVar.F90)

Initialize the following time varying variables:

water : h2osno, h2ocan, h2osoi_liq, h2osoi_ice, h2osoi_vol

snow : snowdp, snowage, snl, dz, z, zi

temperature: t_soisno, t_veg, t_grnd

Note - h2osoi_vol is needed by clm_soilalb -this is not needed on restart since it is computed before the soil albedo computation is called

Note - remaining variables are initialized by calls to ecosystem dynamics and albedo subroutines.

Method: Initial data is saved to instantaneous initial data files for each of the [maxpatch] subgrid patches for each of the [numland] land points. If a subgrid patch is not active (e.g., 3 patches rather than [maxpatch]), the inactive subgrid patches have data values for the first subgrid patch. This way, as long as the land mask DOES NOT change among runs (i.e., [numland] is the same), an initial data file can be used in numerous experiments even if the surface types (and hence [numpatch]) differ

Author: Mariana Vertenstein

INTERFACE:

```
subroutine iniTimeVar (readini, eccen, obliqr, lambm0 , mvelpp, lis, tile)
```

USES:

```
use lis_module
```

```

use tile_module
use precision
use clm_varder
use clm_varctl, only : clmdrv
use clm_varmap , only : numpatch
use clm_varcon , only : bdsno, istice, istwet, istsoil, denice, denh2o, tfrz, spval
use inicFileMod , only : type_inidat, inicrd, histrd
use shr_sys_mod , only : shr_sys_abort
use spmdMod      , only : masterproc
use time_manager, only : get_nstep, get_curr_calday
#if (defined SPMD)
  use mpishorthand, only : mpicom, mpichar
#endif

```

CONTENTS:

```

if (readini) then
  if ( masterproc ) write (6,*) 'Reading initial data '
  call type_inidat(initype)
  if (trim(initype) == 'INICFILE') then
    call inicrd ()
  else if (trim(initype) == 'HISTFILE') then
    call histrd ()
  else
    call shr_sys_abort('initial data type is limited to INIC or HIST file only')
  endif
  do k = 1,numpatch
    do j = 1,nlevsoi
      clm(k)%h2osoi_vol(j) = clm(k)%h2osoi_liq(j)/(clm(k)%dz(j)*denh2o) &
                                + clm(k)%h2osoi_ice(j)/(clm(k)%dz(j)*denice)
    end do
  end do
else
  if ( masterproc ) write (6,*) 'Setting initial data to non-spun up values'

! =====
! Set snow water
! =====

! NOTE: h2ocan, h2osno, snowdp and snowage has valid values everywhere

do k = 1,numpatch
  if (lis%startcode == 4) then
    clm(k)%h2ocan = 0.
    clm(k)%snowage = 0.
  else
    clm(k)%h2ocan = 0.
    if (clm(k)%itypwat == istice) then
      clm(k)%h2osno = 1000.
    end if
  end if
end do

```

```

        else
            clm(k)%h2osno = clmdrv%clm2_iscv
        endif
        clm(k)%snowdp = clm(k)%h2osno/bdsno
        clm(k)%snowage = 0.
    endif
end do

! =====
! Set snow layer number, depth and thickness
! =====
call snowdp2lev ()

! =====
! Set snow/soil temperature
! =====

! NOTE:
! t$-_soisno only has valid values over non-lake
! t$-_lake only has valid values over lake
! t$-_grnd has valid values over all land
! t$-_veg has valid values over all land

do k =1,numpatch
    clm(k)%t_soisno(-nlevsno+1:nlevsoi) = 0
    if (lis%o%startcode == 4) then
        clm(k)%t_veg = clm(k)%forc_t
    else
        clm(k)%t_veg = clmdrv%clm2_it
    endif
    if (.not. clm(k)%lakpoi) then !not lake
        clm(k)%t_soisno(-nlevsno+1:0) = spval
        if (clm(k)%snl < 0) then !snow layer temperatures
            do i = clm(k)%snl+1, 0
                if (lis%o%startcode == 4) then
                    if (clm(k)%forc_t < 273.16) then
                        clm(k)%t_soisno(i) = clm(k)%forc_t
                    else
                        clm(k)%t_soisno(i) = 273.16 - 1.
                    endif
                else
                    if (clmdrv%clm2_it < 273.16) then
                        clm(k)%t_soisno(i) = clmdrv%clm2_it
                    else
                        clm(k)%t_soisno(i) = 273.16 - 1.
                    endif
                endif
            enddo
        endif
    endif
enddo

```

```

        endif
        do i = 1, nlevsoi
            if (lis%o%startcode == 4) then
                if (clm(k)%itypwat == istice) then
                    clm(k)%t_soisno(i) = clm(k)%forc_t
                else if (clm(k)%itypwat == istwet) then
                    clm(k)%t_soisno(i) = clm(k)%forc_t
                else
                    clm(k)%t_soisno(i) = clm(k)%forc_t
                endif
            else
                if (clm(k)%itypwat == istice) then
                    clm(k)%t_soisno(i) = clmdrv%clm2_it
                else if (clm(k)%itypwat == istwet) then
                    clm(k)%t_soisno(i) = clmdrv%clm2_it
                else
                    clm(k)%t_soisno(i) = clmdrv%clm2_it
                endif
            endif
        enddo
        clm(k)%t_grnd = clm(k)%t_soisno(clm(k)%snl+1)
    else
        !lake
        if (lis%o%startcode == 4) then
            clm(k)%t_grnd = clm(k)%forc_t
        else
            clm(k)%t_grnd = clmdrv%clm2_it
        endif
    endif
end do

! =====
! Set snow/soil ice and liquid mass
! =====

! volumetric water is set first and liquid content and ice lens are
! then obtained
! NOTE: h2osoi$-$vol, h2osoi$-$liq and h2osoi$-$ice only have valid values
! over soil
do k = 1,numpatch
    clm(k)%h2osoi_vol(           1:nlevsoi) = spval
    clm(k)%h2osoi_liq(-nlevsno+1:nlevsoi) = spval
    clm(k)%h2osoi_ice(-nlevsno+1:nlevsoi) = spval

    if (.not. clm(k)%lakpoi) then !not lake
        if (clm(k)%snl < 0) then !snow
            do i = clm(k)%snl+1, 0
                clm(k)%h2osoi_ice(i) = clm(k)%dz(i)*250.
                clm(k)%h2osoi_liq(i) = 0.
        endif
    endif
end do

```

```

        enddo
    endif
    do i = 1, nlevsoi           !soil layers
        if (clm(k)%t_soisno(i) <= tfrz) then
            if (lis%o%startcode == 4) then
            else
                clm(k)%h2osoi_ice(i) = clm(k)%dz(i)* &
                    clmdrv%clm2_ism*clm(k)%watsat(i)*denice
            endif
            clm(k)%h2osoi_liq(i) = 0.
            if (clm(k)%itypwat==istwet .or. clm(k)%itypwat==istice) &
                clm(k)%h2osoi_ice(i)=clm(k)%dz(i)*denice
        else
            if (lis%o%startcode == 4) then
                print*, 'Not supposed to be called..'
            else
                clm(k)%h2osoi_liq(i) = clm(k)%dz(i)* &
                    clmdrv%clm2_ism*clm(k)%watsat(i)*denh2o
            endif
            clm(k)%h2osoi_ice(i) = 0.
            if (clm(k)%itypwat==istwet .or. clm(k)%itypwat==istice) &
                clm(k)%h2osoi_liq(i)=clm(k)%dz(i)*denh2o
            endif
        endif
    enddo

    do i = 1,nlevsoi
        if (clm(k)%itypwat == istsoil) then
            clm(k)%h2osoi_vol(i) = 0.3_r8
            clm(k)%h2osoi_vol(i) = clm(k)%h2osoi_liq(i)/&
                (clm(k)%dz(i)*denh2o) &
                + clm(k)%h2osoi_ice(i)/(clm(k)%dz(i)*denice)
        else
            clm(k)%h2osoi_vol(i) = 1.0_r8
        endif
        clm(k)%h2osoi_vol(i) = min(clm(k)%h2osoi_vol(i),clm(k)%watsat(i))
    end do
    endif
end do
end if ! end of arbitrary initialization if-block

! =====
! Remaining variables are initialized by calls to ecosystem dynamics and
! albedo subroutines.
! Note: elai, esai, frac_veg_nosno are computed in Ecosysdyn and needed
! by Fwet and SurfaceAlbedo
! Note: fwet is needed in routine clm_twostream (called by clm_surfalb)
! =====

```

```

if ( masterproc ) then
    calday = get_curr_calday()
endif
#if ( defined OPENDAP )
    call MPI_BCAST(calday,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
#endif
doalb = .true.

print*, 'DBG: iniTimeVar -- calling clm2lairead',' (',iam,')'
call clm2lairead(lis,tile)

#if (defined DGVM)
    call iniTimeConstDGVM()
#endif

if ( masterproc ) then
    tmp_nstep = get_nstep()
endif
#if ( defined OPENDAP )
    call MPI_BCAST(tmp_nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
#endif
do k = 1,numpatch
    clm(k)%nstep = tmp_nstep
    call EcosystemDyn (clm(k), doalb, .false.)
    clm(k)%frac_sno = clm(k)%snowdp/(0.1 + clm(k)%snowdp)
    clm(k)%frac_veg_nosno = clm(k)%frac_veg_nosno_alb
    call Fwet(clm(k))
    call SurfaceAlbedo (clm(k), calday, eccen, obliqr, lambm0, mvelpp)
end do
return

end subroutine iniTimeVar
!----- ! NASA GSFC Land Information
Systems LIS 2.3 ! -----

```

ROUTINE : readclm2crd.F90

Routine to read CLM specific parameters from the card file.

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Code

INTERFACE:

subroutine readclm2crd(clmdrv)

USES:

use clm2drv_module

CONTENTS:

```

open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=clm2)
print*, 'Running CLM2 LSM:'
print*, 'CLM2 Active Restart File: ', clmdrv%CLM2_RFILE
clmdrv%clm2open=0
close(11)

```

ROUTINE : tile2clm2.F90

This subroutine equates tile space variables between standard LDAS and that used in CLM2

REVISION HISTORY:

7 Nov 2002: Jon Gottschalck; Initial code

INTERFACE:

SUBROUTINE TILE2CLM2 (LIS,GRID,GINDEX,WXTY,VEGXY)

USES:

```

use clm_varder      ! CLM2 tile variables
use clm_varctl, only : nsrest, fpftcon
use lis_module       ! LIS variables
use grid_module      ! Grid variables
#ifndef OPENDAP
use opendap_module, only : parm_nc, parm_nr
#endif

```

1.35.14 mapvegc.F90 (Source File: mapvegc.F90)

This subroutine converts the UMD classes to the SIB classes used by NOAH LSM (v 2.5).
(Originally from Dag Lohmann at NCEP)

REVISION HISTORY:

28 Apr 2002, K Arsenault: Added NOAH LSM to LDAS.

INTERFACE:

```
subroutine mapvegc(vegt)
```

CONTENTS:

```

!-----
! Convert UMD Classes to SIB Classes.
!-----
if (vegt .eq. 1) sibveg = 4
if (vegt .eq. 2) sibveg = 1

```

```

if (vegt .eq. 3) sibveg = 5
if (vegt .eq. 4) sibveg = 2
if (vegt .eq. 5) sibveg = 3
if (vegt .eq. 6) sibveg = 3
if (vegt .eq. 7) sibveg = 6
if (vegt .eq. 8) sibveg = 8
if (vegt .eq. 9) sibveg = 9
if (vegt .eq. 10) sibveg = 7
if (vegt .eq. 11) sibveg = 12
if (vegt .eq. 12) sibveg = 11
if (vegt .eq. 13) sibveg = 11
if (vegt .gt. 13) then
    sibveg = 7
end if
vegt=sibveg
return

```

1.35.15 noah_alb.F90: (Source File: noah_alb.F90)

This subroutine takes quarterly surface albedo (snow-free) data and day to interpolate and determine the actual value of the albedo for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the dates of January 31, April 30, July 31, and October 31.

REVISION HISTORY:

28 Apr 2002: K. Arsenault; Added NOAH LSM to LDAS, initial code

INTERFACE:

```
subroutine noah_alb
```

USES:

```

use time_module
use noah_varder      ! NOAH tile variables
use time_manager
use lisdrv_module, only : grid,tile,lis
#if ( defined OPENDAP )
    use opendap_module
#endif
implicit none

```

CONTENTS:

```

zeroi=0
noahdrv%noah_aflag = 0
!-----
```

```
! Determine Dates of the quarters in terms of Year (e.g., 1999.3)
```

```
!---
```

```
    time=lis%t%time
    yr=lis%t%yr
```

```
!---
    ! January 31
!---
```

```
    janda=31
    janmo=01
    call date2time(jan31,doy1,gmt1,year,janmo,&
                   janda,zeroi,zeroi,zeroi)
```

```
!---
    ! April 30
!---
```

```
    aprda=30
    aprmo=04
    call date2time(apr30,doy1,gmt1,year,aprmo,&
                   aprda,zeroi,zeroi,zeroi)
```

```
!---
    ! July 31
!---
```

```
    julda=31
    julmo=07
    call date2time(jul31,doy1,gmt1,year,julmo,&
                   julda,zeroi,zeroi,zeroi)
```

```
!---
    ! October 31
!---
```

```
    octda=31
    octmo=10
    call date2time(oct31,doy1,gmt1,year,octmo,&
                   octda,zeroi,zeroi,zeroi)
```

```
!---
    ! Determine which two quarterly albedo files book-end model time.
```

```
!---
```

```
if ( time.ge.jan31 .and. time.le.apr30 ) then
  qq1="01"
  qq2="02"
  qdif = apr30-jan31
  timdif = time-jan31
  albflag = 1
elseif ( time.ge.apr30 .and. time.le.jul31 ) then
  qq1="02"
  qq2="03"
  qdif = jul31-apr30
  timdif = time-apr30
  albflag = 2
```

```

elseif ( time.ge.jul31 .and. time.le.oct31 ) then
  qq1="03"
  qq2="04"
  qdif = oct31-jul31
  timdif = time-jul31
  albflag = 3
elseif ( time.ge.oct31 ) then
  qq1="04"
  qq2="01"
  qdif = (jan31+1.0)-oct31
  timdif = time-oct31
  albflag = 4
elseif ( time.lt.jan31) then
  qq1="04"
  qq2="01"
  oct31=oct31-1.0
  qdif = jan31-oct31
  timdif = time-oct31
  albflag = 5
endif

if(noahdrv%noah_albtime .ne. albflag) then
  noahdrv%noah_albtime = albflag
  noahdrv%noah_aflag = 1
!-----
! Open the needed two quarterly snow-free albedo files
!-----


#if ( defined OPENDAP )
  print*, 'MSG: noah_alb -- Retrieving ALBEDO file ',&
    trim(noahdrv%noah_albfile)//'albedo_//QQ1//'.bfsa',&
    ',iam,)'
  call system("opendap_scripts/getalbedo.pl "//ciam// " // &
    trim(noahdrv%noah_albfile)//'albedo_//QQ1//'.bfsa' &
    // " //cparm_slat// " //cparm_nlat &
    // " //cparm_wlon// " //cparm_elon// " //QQ1)
  print*, 'MSG: noah_alb -- Retrieving ALBEDO file ', &
    trim(noahdrv%noah_albfile)//'albedo_//QQ2//'.bfsa',&
    ',iam,)'
  call system("opendap_scripts/getalbedo.pl "//ciam// " // &
    trim(noahdrv%noah_albfile)//'albedo_//QQ2//'.bfsa' &
    // " //cparm_slat// " //cparm_nlat &
    // " //cparm_wlon// " //cparm_elon// " //QQ2)
#endif
  print*, 'MSG: noah_alb -- Retrieving ALBEDO file ',&
    trim(noahdrv%noah_albfile)//'albedo_//QQ1//'.bfsa',&
    ',iam,)'
OPEN (10, &

```

```

FILE=trim(NOAHDRV%NOAH_ALBFILE)//'albedo_ '//QQ1//'.bfsa',&
STATUS='OLD',FORM='UNFORMATTED')
print*, 'MSG: noah_alb -- Retrieving ALBEDO file ', &
trim(noahdrv%noah_albfile)//'albedo_ '//QQ2//'.bfsa',&
' (',iam,')
OPEN (11, &
FILE=trim(NOAHDRV%NOAH_ALBFILE)//'albedo_ '//QQ2//'.bfsa',&
STATUS='OLD',FORM='UNFORMATTED')

read(10) value1
read(11) value2
close(10)
close(11)

!-----
! Assign quarterly albedo fractions to each tile.
!-----

do i=1,lis%d%nch
  if((value1(tile(i)%col, tile(i)%row-tnroffset).ne.-9999.000) &
     .and. (value2(tile(i)%col,tile(i)%row-tnroffset)&
     .ne.-9999.000)) then
    noah(i)%albsf1= value1(tile(i)%col, tile(i)%row-tnroffset)
    noah(i)%albsf2= value2(tile(i)%col, tile(i)%row-tnroffset)
  endif
enddo
endif      ! End albfflag selection
!-----
! Assign albedo fractions to each tile and interpolate daily.
!-----

if (noahdrv%noah_albdchk .ne. lis%t%da) then
  noahdrv%noah_aflag = 1
  do i=1,lis%d%nch
    if (noah(i)%albsf1 .ne. -9999.000) then
      valdif(i) = noah(i)%albsf2 - noah(i)%albsf1
      noah(i)%albsf = (timdif*valdif(i)/qdif)+noah(i)%albsf1
    endif
  end do
  noahdrv%noah_albdchk=lis%t%da

  if(lis%o%wparam.eq.1) then
    allocate(albout(lis%d%lnc,lis%d%lnr))
    do i=1,lis%d%nch
      if(grid(i)%lat*1000.ge.lis%d%kgds(4).and. &
         grid(i)%lat*1000.le.lis%d%kgds(7).and. &
         grid(i)%lon*1000.ge.lis%d%kgds(5).and. &
         grid(i)%lon*1000.le.lis%d%kgds(8)) then
        rindex = tile(i)%row - (lis%d%kgds(4)-lis%d%kgds(44)) &
                  /lis%d%kgds(9)
        cindex = tile(i)%col - (lis%d%kgds(5)-lis%d%kgds(45)) &

```

```

        /lis%d%kgds(10)
        albout(cindex,rindex) = noah(i)%albsf
    endif
enddo
open(32,file="albout.bin",form='unformatted')
write(32) albout
close(32)
deallocate(albout)
end if
endif ! End daily interpolation
return

```

1.35.16 noah_almaout.F90 (Source File: noah_almaout.F90)

LIS NOAH data writer: Binary and stat files in ALMA convention

REVISION HISTORY:

4 Nov. 1999: Jon Radakovich; Initial Code
 28 Apr. 2002: Kristi Arsenault; Added NOAH LSM to LDAS
 15 Jun 2003: Sujay Kumar; ALMA version

INTERFACE:

```
subroutine noah_almaout (ld,tile,gindex)
```

USES:

```

use lis_module      ! LIS non-model-specific 1-D variables
use tile_module    ! LIS non-model-specific tile variables
use grid_module    ! LIS non-model-specific grid variables
use noah_varder   ! NOAH-specific variables
use time_manager, only : get_nstep

implicit none

```

ARGUMENTS:

```

type (lisdec) LD
type (tiledec) tile(ld%d%glbnch)
integer :: gindex(ld%d%lnc, ld%d%lnr)

```

CONTENTS:

```

!-----
! Test to see if output writing interval has been reached
!-----
if(mod(ld%t%gmt,noahdrv%writeintn).eq.0)then

```

```

noahdrv%numoutnh=noahdrv%numoutnh+1
write(unit=temp1,fmt='(i4,i2,i2)')ld%t%yr,ld%t%mo,ld%t%da
read(unit=temp1,fmt='(8a1)') ftime
do i=1,8
    if(ftime(i).eq.(' '))ftime(i)='0'
enddo
write(unit=temp1,fmt='(i4)')ld%t%yr
read(unit=temp1,fmt='(8a1)')ftimec
do i=1,4
    if(ftimec(i).eq.(' '))ftimec(i)='0'
enddo
write(unit=temp1,fmt='(a6,i3,a1)') '/LIS.E',ld%o%expcode,'.'
read(unit=temp1,fmt='(80a1)') (fname(i),i=1,11)
do i=1,11
    if(fname(i).eq.(' '))fname(i)='0'
enddo
write(unit=temp1,fmt='(a40)') ld%o%odir
read(unit=temp1,fmt='(40a1)') (fbase(i),i=1,40)
c=0
do i=1,40
    if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
enddo

write(unit=temp1,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)')'/EXP', &
    ld%o%expcode,'/NOAH/', &
    ld%t%yr,'/',ld%t%yr,ld%t%mo,ld%t%da
read(unit=temp1,fmt='(80a1)') (fyrmadir(i),i=1,26)
do i=1,26
    if(fyrmadir(i).eq.(' '))fyrmadir(i)='0'
enddo

write(unit=temp1,fmt='(a9)')'mkdir -p '
read(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9)

write(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
    (fyrmadir(i),i=1,26)
read(unit=temp1,fmt='(a80)')mkfyrmo
call system(mkfyrmo)
!-----
! Generate file name for BINARY output
!-----
if(ld%o%wout.eq.1) then
    write(unit=fbinname, fmt='(i4,i2,i2,i2)') ld%t%yr,ld%t%mo, &
        ld%t%da,ld%t%hr
    read(unit=fbinname,fmt='(10a1)') ftimeb
    do i=1,10
        if(ftimeb(i).eq.(' '))ftimeb(i)='0'
    enddo

```

```

        write(unit=fbinname,fmt='(a9)') '.NOAHgbin'
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,9)

        write(unit=fbinname,fmt='(80a1)')(fbase(i),i=1,c), &
            (fyrmmdir(i),i=1,26), &
            (fname(i),i=1,11),(ftimeb(i),i=1,10), &
            (fsubgb(i),i=1,9)
        read(unit=fbinname,fmt='(a80)')filengb
!-----
! Open statistical output file
!-----
      if(noahdrv%noahopen.eq.0)then
        file='Noahstats.dat'
        call openfile(name,ld%o%odir,ld%o%expcode,file)
        if(ld%o%startcode.eq.1)then
          open(65,file=name,form='formatted',status='unknown', &
              position='append')
        else
          open(65,file=name,form='formatted',status='replace')
        endif
        noahdrv%noahopen=1
      endif

      write(65,996)' Statistical Summary of Noah output for: ', &
          ld%t%mo,'/',ld%t%da,'/',ld%t%yr,ld%t%hr,:',ld%t%mn,:',ld%t%ss
996    format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
      write(65,*)
      write(65,997)
997    format(t27,'Mean',t41,'Stdev',t56,'Min',t70,'Max')
      endif
      if(ld%o%wout.eq.1) then
        open(58,file=filengb,form='unformatted')
      endif
      if(ld%o%wout.eq.1) then
        if(ld%o%wtil.eq.1) then
          call noah_tileout(ld,tile,58,65)
        else
          call noah_gridout(ld,tile,58,65)
        endif
      endif
      noah%count=0 !reset counters

      write(65,*)
      write(65,*)
    endif

```

1.35.17 noah_atmdrv.F90 (Source File: noah_atmdrv.F90)

Transfer forcing from grid to tile space.

REVISION HISTORY:

15 Oct 1999: Paul Houser; Initial Code

28 Jan 2002: Jon Gottschalck; Added option for different number of forcing variables

INTERFACE:

```
subroutine noah_f2t(t, forcing)
```

USES:

```
use lisdrv_module , only : lis
use spmdMod
use tile_spmdMod
use noah_varder
```

CONTENTS:

```
do f=1,lis%f%nforce
    noah(t)%forcing(f)=forcing(f)
enddo
```

1.35.18 noah_coldstart.F90 (Source File: noah_coldstart.F90)

Routine for noah initialization from cold start

INTERFACE:

```
subroutine noah_coldstart()
```

USES:

```
use lisdrv_module, only: lis
use noah_varder
use time_module
use spmdMod, only: iam
```

CONTENTS:

```
if ( lis%o%startcode == 2 ) then
    print*, 'MSG: noah_coldstart -- cold-starting noah', &
        '...using ics from card file', (', iam, ')
    print*, 'DBG: noah_coldstart -- nch', lis%d%nch, &
        (', iam, ')
    do t=1,lis%d%nch
        noah(t)%t1=noahdrv%noah_it
```

```

noah(t)%t1=280.0
noah(t)%cmc=0.0004
noah(t)%snowh=0.0
noah(t)%sneqv=0.0
noah(t)%ch=0.0150022404
noah(t)%cm=0.0205970779
do l=1,4
    noah(t)%stc(l)=noahdrv%noah_it
enddo
noah(t)%smc(1)=0.3252287
noah(t)%smc(2)=0.3194746
noah(t)%smc(3)=0.3172167
noah(t)%smc(4)=0.3078052
noah(t)%sh2o(1)=0.1660042
noah(t)%sh2o(2)=0.2828006
noah(t)%sh2o(3)=0.3172163
noah(t)%sh2o(4)=0.3078025
enddo
lis%t%yr=lis%t%syr
lis%t%mo=lis%t%smo
lis%t%da=lis%t%sda
lis%t%hr=lis%t%shr
lis%t%mn=lis%t%smn
lis%t%ss=lis%t%sss

call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr,&
              lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
write(*,*)'MSG: noah_coldstart -- Using lis.crd start time ',&
           lis%t%time, ' (', iam, ')'
write(79,*)'MSG: noah_coldstart -- Using lis.crd start time ',&
           lis%t%time, ' (', iam, ')'
endif

```

1.36 Fortran: Module Interface noahdrv_module.F90 (Source File: noahdrv_module.F90)

Module for runtime specific Noah variables

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module noahdrv_module
```

ARGUMENTS:

```

type noahdrvdec
    integer :: noahopen          !Keeps track of opening files
    integer :: numouthn          !Counts number of output times for Noah
    integer :: noah_nvegp         !Number of static vegetation parameter
    integer :: noah_nsoilp         !Number of static soil parameters
    integer :: noah_zst           !Number of Zobler soil classes
    integer :: noah_gflag          !Time flag to update gfrac files
    integer :: noah_albtime        !Time flag to update albedo files
    integer :: noah_aflag           !Time flag to update albedo files
    integer :: noah_albdchk        !Day check to interpolate alb values
    integer :: noah_gfracdchk      !Day check to interpolate gfrac value
    character*40 :: NOAH_RFILE !NOAH Active Restart File
    character*40 :: NOAH_MFILE !NOAH model init. restart file
    CHARACTER*40 :: NOAH_VFILE !NOAH Static Vegetation Parameter File
    CHARACTER*40 :: NOAH_SFILE !NOAH Soil Parameter File
    CHARACTER*40 :: NOAH_MGFILE !NOAH Monthly Veg. Green Frac.
    CHARACTER*40 :: NOAH_ALBFILE !NOAH Quart. Snow-free albedo
    CHARACTER*50 :: NOAH_MXSNAL !NOAH GLDAS max snow albedo
    CHARACTER*50 :: NOAH_TBOT     !NOAH GLDAS Bottom Temp
    REAL*8 :: NOAH_GFRACTIME   !Time flag to update gfrac files
    REAL :: NOAH_ISM            !NOAH Initial Soil Moisture (m3/m3)
    REAL :: NOAH_IT              !NOAH Initial Soil Temperature (K)
    REAL :: WRITEINTN           !NOAH Output Interval (hours)
end type noahdrvdec

```

1.36.1 noah_dynsetup.F90 (Source File: noah_dynsetup.F90)

Updates the time dependent NOAH variables

REVISION HISTORY:

15 Apr 2002: Sujay Kumar Initial Specification

INTERFACE:

```
subroutine noah_dynsetup()
```

USES:

```

use lisdrv_module, only: lis,tile
use noah_varder
use spmdMod, only : masterproc, npes
use noahpardef_module

```

CONTENTS:

```

integer :: t, n,ier
#if ( ! defined OPENDAP )

```

```

if ( npes > 1 ) then
    call noah_gather
endif
if(masterproc) then
#endifif
    call noah_gfrac()
    call noah_alb()
#ifndefif ( ! defined OPENDAP )
endif
call MPI_BCAST(noahdrv%noah_gflag,1,MPI_INTEGER,0, &
    MPI_COMM_WORLD,ierr)
call MPI_BCAST(noahdrv%noah_aflag,1,MPI_INTEGER,0, &
    MPI_COMM_WORLD,ierr)
if( npes > 1 .and. ( noahdrv%noah_gflag==1 .or. &
    noahdrv%noah_aflag ==1 ) ) then
    call noah_scatter
endif
#endifif

```

1.36.2 noah_gather.F90 (Source File: noah_gather.F90)

Gathers noah tiles

REVISION HISTORY:

Apr 2003 ; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine noah_gather()
```

USES:

```

use tile_spmdMod
use noah_varder
use noahpardef_module
```

CONTENTS:

```

#ifndefif (defined SPMD)
call MPI_GATHERV(noah(1:di_array(iam)),di_array(iam), &
    MPI_NOAH_STRUCT,noah,di_array,displs,MPI_NOAH_STRUCT, &
    0,MPI_COMM_WORLD, ierr)
#endifif

```

1.36.3 noah_gfrac.F90 (Source File: noah_gfrac.F90)

This subroutine takes vegetation greenness fraction data and the date to interpolate and determine the actual value of the greenness fraction for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the 16th of the given month, at 00Z.

REVISION HISTORY:

28 Apr 2002: K. Arsenault; Added NOAH LSM to LDAS, initial code

INTERFACE:

```
subroutine noah_gfrac
```

USES:

```
use noah_varder
use time_manager
use time_module
use lisdrv_module, only : grid,tile,lis
#if ( defined OPENDAP )
  use opendap_module
#endif
```

CONTENTS:

```
noahdrv%noah_gflag = 0
zeroi=0
numi=16
!-----
! Determine Monthly data Times (Assume Monthly value valid at DA=16)
!-----
if(lis%t%da.lt.16)then
  mo1=lis%t%mo-1
  yr1=lis%t%yr
  if(mo1.eq.0)then
    mo1=12
    yr1=lis%t%yr-1
  endif
  mo2=lis%t%mo
  yr2=lis%t%yr
else
  mo1=lis%t%mo
  yr1=lis%t%yr
  mo2=lis%t%mo+1
  yr2=lis%t%yr
  if(mo2.eq.13)then
    mo2=1
    yr2=lis%t%yr+1
  endif
```

```

        endif

        call date2time(time1,doy1,gmt1,yr1,mo1,&
                      numi,zeroi,zeroi,zeroi)
        call date2time(time2,doy2,gmt2,yr2,mo2,&
                      numi,zeroi,zeroi,zeroi)

!-----
!  Weights to be used to interpolate greenness fraction values.
!-----
        wt1= (time2-lis%t%time)/(time2-time1)
        wt2= (lis%t%time-time1)/(time2-time1)

!-----
!  Determine if GFRAC files need to be updated
!-----
        if(time2 .gt. noahdrv%noah_gfractime) then
            gfrac_flag = 1
        else
            gfrac_flag = 0
        endif

        if(gfrac_flag .eq. 1) then
            noahdrv%noah_gfractime = time2
            noahdrv%noah_gflag = 1
        end if

!-----
! Open greenness fraction dataset of months corresponding to
! time1 and time2 for selected LDAS domain and read data.
!-----
        write(mmm1,3) mo1
        write(mmm2,3) mo2
3      format(i2.2)

#if ( defined opendap )
    print*, 'msg: noah_gfrac -- retrieving gfrac file ',&
              trim(noahdrv%noah_mgfile)//'gfrac_//mm1//'.bfsa',&
              ',(,iam,)'
    call system("opendap_scripts/getgfrac.pl "//ciam//" //&
                trim(noahdrv%noah_mgfile)//'gfrac_//mm1//'.bfsa'      &
                //" //cparm_slat//" //cparm_nlat                  &
                //" //cparm_wlon//" //cparm_elon//" //mm1)
    print*, 'msg: noah_gfrac -- retrieving gfrac file ', &
              trim(noahdrv%noah_mgfile)//'gfrac_//mm2//'.bfsa',&
              ',(,iam,)'
    call system("opendap_scripts/getgfrac.pl "//ciam//" //&
                trim(noahdrv%noah_mgfile)//'gfrac_//mm2//'.bfsa'      &
                //" //cparm_slat//" //cparm_nlat                  &
                //" //cparm_wlon//" //cparm_elon//" //mm2)
#endif

```

```

print*, 'msg: noah_gfrac -- retrieving gfrac file ',&
trim(noahdrv%noah_mgfile)//'gfrac_//mm1//'.bfsa',&
' (',iam,')'
open (10, &
      file=trim(noahdrv%noah_mgfile)//'gfrac_//mm1//'.bfsa', &
      status='old', form='unformatted')
print*, 'msg: noah_gfrac -- retrieving gfrac file ', &
trim(noahdrv%noah_mgfile)//'gfrac_//mm2//'.bfsa',&
' (',iam,')'
open (11, &
      file=trim(noahdrv%noah_mgfile)//'gfrac_//mm2//'.bfsa', &
      status='old', form='unformatted')

read(10) value1
read(11) value2
close(10)
close(11)

!-----
! Assign MONTHLY vegetation greenness fractions to each tile.
!-----
do i=1,lis%d%nch
  if((value1(tile(i)%col, tile(i)%row-tnroffset) .ne. -9999.000) &
     .and.(value2(tile(i)%col, tile(i)%row-tnroffset).ne.-9999.000)) &
    then
      noah(i)%vegmp1=value1(tile(i)%col, tile(i)%row-tnroffset)
      noah(i)%vegmp2=value2(tile(i)%col, tile(i)%row-tnroffset)
    endif
  end do
endif
!-----
! Interpolate greenness fraction values once daily
!-----

if (noahdrv%noah_gfracdchk .ne. lis%t%da) then
  noahdrv%noah_gflag = 1
  do i=1,lis%d%nch
    noah(i)%vegip = (wt1*noah(i)%vegmp1)+(wt2*noah(i)%vegmp2)
  end do
  noahdrv%noah_gfracdchk = lis%t%da
  print*, 'Done noah_gfrac', (',iam,')'

  if(lis%o%wparam.eq.1) then
    allocate(gfracout(lis%d%lnc,lis%d%lnr))
    do i=1,lis%d%nch
      if(grid(i)%lat*1000.ge.lis%d%kgds(4).and. &
         grid(i)%lat*1000.le.lis%d%kgds(7).and. &
         grid(i)%lon*1000.ge.lis%d%kgds(5).and. &

```

```

        grid(i)%lon*1000.le.lis%d%kgds(8)) then
        rindex = tile(i)%row - (lis%d%kgds(4)-lis%d%kgds(44)) &
        /lis%d%kgds(9)
        cindex = tile(i)%col - (lis%d%kgds(5)-lis%d%kgds(45)) &
        /lis%d%kgds(10)
        gfracout(cindex,rindex) = noah(i)%vegip*1.0
    endif
enddo
open(32,file="gfracout.bin",form='unformatted')
write(32) gfracout
close(32)
deallocate(gfracout)
endif
end if
return

```

1.36.4 noah_gridout.F90 (Source File: noah_gridout.F90)

LIS NOAH data writer: Writes noah output in grid space

REVISION HISTORY:

02 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
subroutine noah_gridout(ld,tile,ftn,ftn_stats)
```

USES:

```

use lis_module
use tile_module
use noah_varder

implicit none

type(lisdec) :: ld
type(tiledec) :: tile
integer :: ftn,ftn_stats

```

CONTENTS:

```

do t=1,ld%d%glbnch
    if(noah(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = noah(t)%forcing(8)
    else

```

```

rainf(t) = noah(t)%forcing(8)
snowf(t) = 0.0
endif
enddo
!-----
! General Energy Balance Components
!-----
noah%swnet = noah%swnet/float(noah%count)
call tile2grid(noah%swnet,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Net shortwave radiation (surface) (W/m2)

call stats(noah%swnet,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SWnet(W/m2)', &
                     vmean,vstdev,vmin,vmax

noah%lwnet = (-1)*noah%lwnet/float(noah%count)
call tile2grid(noah%lwnet,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Net longwave radiation (surface) (W/m2)

call stats(noah%lwnet,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'LWnet(W/m2)',&
                     vmean,vstdev,vmin,vmax

noah%qle = noah%qle/float(noah%count)
call tile2grid(noah%qle,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Latent Heat Flux (W/m2)

call stats(noah%qle,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'Qle(W/m2)',&
                     vmean,vstdev,vmin,vmax

noah%qh = noah%qh/float(noah%count)
call tile2grid(noah%qh,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Sensible Heat Flux (W/m2)

call stats(noah%qh,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'Qh(W/m2)',&
                     vmean,vstdev,vmin,vmax

noah%qg = noah%qg/float(noah%count)
call tile2grid(noah%qg,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Ground Heat Flux (W/m2)

call stats(noah%qg,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)

```

```

      vstdev,vmin, vmax)
write(ftn_stats,999) 'Qg(W/m2)',&
      vmean,vstdev,vmin,vmax
!-----
! General Water Balance Components
!-----
noah%snowf = noah%snowf/float(noah%count)
call tile2grid(noah%snowf,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp !Snowfall rate (kg/m2s)

call stats(noah%snowf,1d%d%udef,1d%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,998) 'Snowf(kg/m2s)',&
      vmean,vstdev,vmin,vmax

noah%rainf = noah%rainf/float(noah%count)
call tile2grid(noah%rainf,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp !Snowfall rate (kg/m2s)

call stats(noah%rainf,1d%d%udef,1d%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,998) 'Rainf(kg/m2s)',&
      vmean,vstdev,vmin,vmax

noah%evap = noah%evap/float(noah%count)
call tile2grid(noah%evap,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp !Evapotranspiration (kg/m2s)???

call stats(noah%evap,1d%d%udef,1d%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,998) 'Evap(kg/m2s)',&
      vmean,vstdev,vmin,vmax

noah%qs = noah%qs/float(noah%count)
call tile2grid(noah%qs,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp !Surface Runoff(kg/m2s)

call stats(noah%qs,1d%d%udef,1d%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,998) 'Qs(kg/m2s)',&
      vmean,vstdev,vmin,vmax

noah%qsb = noah%qsb/float(noah%count)
call tile2grid(noah%qsb,gtmp,1d%d%glbnch,1d%d%lnc,1d%d%lnr,tile)
write(ftn) gtmp !Subsurface Runoff (kg/m2s)???

call stats(noah%qsb,1d%d%udef,1d%d%glbnch,vmean, &
      vstdev,vmin, vmax)

```

```

write(ftn_stats,998) 'Qsb(kg/m2s)',&
     vmean,vstdev,vmin,vmax

noah%qsm = noah%qsm/float(noah%count)
call tile2grid(noah%qsm,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gttmp !Snowmelt (kg/m2s)

call tile2grid(noah%smc(1)*1000.0*0.1+ &
    noah%smc(2)*1000.0*0.3 + &
    noah%smc(3)*1000.0*0.6 + &
    noah%smc(4)*1000.0 -noah%soilm_prev, &
    gttmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gttmp      !DelSoilMoist
call stats(noah%smc(1)*1000.0*0.1+ &
    noah%smc(2)*1000.0*0.3 + &
    noah%smc(3)*1000.0*0.6 + &
    noah%smc(4)*1000.0 -noah%soilm_prev, &
    ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,999) 'DelSoilMoist(kg/m2s)', &
     vmean,vstdev,vmin,vmax
call tile2grid( noah%sneqv*1000.0-noah%swe_prev,gttmp,&
    ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gttmp      !DelSWE
call stats( noah%sneqv*1000.0-noah%swe_prev, &
    ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,999) 'DelSWE(kg/m2s)', &
     vmean,vstdev,vmin,vmax
!-----
! Surface State Variables
!-----

call tile2grid(noah%avgsurft,gttmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gttmp !Average Surface Temperature (K)

call stats(noah%avgsurft,ld%d%udef,ld%d%glbnch,vmean, &
    vstdev,vmin, vmax)
write(ftn_stats,999) 'AvgSurfT(K)',&
     vmean,vstdev,vmin,vmax

call tile2grid(noah%albedo,gttmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gttmp !Surface Albedo (-)

call stats(noah%albedo,ld%d%udef,ld%d%glbnch,vmean, &
    vstdev,vmin, vmax)
write(ftn_stats,998) 'Albedo(-)',&
     vmean,vstdev,vmin,vmax

noah%swe= noah%swe/float(noah%count)

```

```

call tile2grid(noah%swe,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp !Snow water equivalent (kg/m2)

call stats(noah%swe,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'SWE(kg/m2)',&
           vmean,vstdev,vmin,vmax
!-----
! Subsurface State Variables
!-----
noah%soilmoist1= noah%soilmoist1/float(noah%count)
call tile2grid(noah%soilmoist1,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Soil water content for layer1 (kg/m2)

call stats(noah%soilmoist1,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist1(kg/m2)',&
           vmean,vstdev,vmin,vmax

noah%soilmoist2= noah%soilmoist2/float(noah%count)
call tile2grid(noah%soilmoist2,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Soil water content for layer2 (kg/m2)

call stats(noah%soilmoist2,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist2(kg/m2)',&
           vmean,vstdev,vmin,vmax

noah%soilmoist3= noah%soilmoist3/float(noah%count)
call tile2grid(noah%soilmoist3,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Soil water content for layer3 (kg/m2)

call stats(noah%soilmoist3,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist3(kg/m2)',&
           vmean,vstdev,vmin,vmax

noah%soilmoist4= noah%soilmoist4/float(noah%count)
call tile2grid(noah%soilmoist4,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Soil water content for layer4 (kg/m2)

call stats(noah%soilmoist4,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist4(kg/m2)',&
           vmean,vstdev,vmin,vmax

noah%soilwet= noah%soilwet/float(noah%count)
call tile2grid(noah%soilwet,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)

```

```

write(ftn) gtmp ! Total Soil Wetness (-)

call stats(noah%soilwet,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'SoilWet(-)',&
           vmean,vstdev,vmin,vmax
!-----
! Evaporation Components
!-----
noah%tveg= noah%tveg/float(noah%count)
call tile2grid(noah%tveg,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Vegetation transpiration (kg/m2s)

call stats(noah%tveg,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'TVeg(kg/m2s)',&
           vmean,vstdev,vmin,vmax

noah%esoil= noah%esoil/float(noah%count)
call tile2grid(noah%esoil,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Bare soil evaporation (kg/m2s)

call stats(noah%esoil,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'ESoil(kg/m2s)',&
           vmean,vstdev,vmin,vmax

noah%rootmoist = noah%rootmoist/float(noah%count)
call tile2grid(noah%rootmoist,gtmp,ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp ! Root zone soil moisture (kg/m2)

call stats(noah%rootmoist,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'RootMoist(kg/m2)',&
           vmean,vstdev,vmin,vmax

if(ld%o%wfor.eq.1) then
  call tile2grid(sqrt(noah%forcing(5)*noah%forcing(5)+ &
                     noah%forcing(6)*noah%forcing(6)),gtmp, &
                     ld%d%glbnch,ld%d%lnc,ld%d%lnr,tile)
  write(ftn) gtmp      !Wind
  call stats(sqrt(noah%forcing(5)*noah%forcing(5)+ &
                 noah%forcing(6)*noah%forcing(6)), &
                 ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
  write(ftn_stats,999) 'Wind(m/s)', &
           vmean,vstdev,vmin,vmax

  call tile2grid(rainf, &

```

```

        gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !Rainf
call stats(rainf, &
           ld%d%udef,ld%d%glnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,998) 'Rainf(kg/m2s)', &
                     vmean,vstdev,vmin,vmax

call tile2grid(snowf, &
               gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !Snowf
call stats(snowf, &
           ld%d%udef,ld%d%glnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,998) 'Snowf(kg/m2s)', &
                     vmean,vstdev,vmin,vmax

call tile2grid(noah%forcing(1),gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !Tair
call stats(noah%forcing(1),ld%d%udef,ld%d%glnch,vmean,vstdev, &
           vmin, vmax)
write(ftn_stats,999) 'Tair(K)', &
                     vmean,vstdev,vmin,vmax
call tile2grid(noah%forcing(2),gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !Qair
call stats(noah%forcing(2),ld%d%udef,ld%d%glnch,vmean,vstdev, &
           vmin, vmax)
write(ftn_stats,999) 'Qair(kg/kg)', &
                     vmean,vstdev,vmin,vmax
call tile2grid(noah%forcing(7),gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !PSurf
call stats(noah%forcing(7),ld%d%udef,ld%d%glnch,vmean,vstdev, &
           vmin, vmax)
write(ftn_stats,999) 'PSurf(Pa)', &
                     vmean,vstdev,vmin,vmax
call tile2grid(noah%forcing(3),gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !SWdown
call stats(noah%forcing(3),ld%d%udef,ld%d%glnch,vmean,vstdev, &
           vmin, vmax)
write(ftn_stats,999) 'SWdown (W/m2)', &
                     vmean,vstdev,vmin,vmax

call tile2grid(noah%forcing(4),gtmp,ld%d%glnch,ld%d%lnc,ld%d%lnr,tile)
write(ftn) gtmp          !LWdown
call stats(noah%forcing(4),ld%d%udef,ld%d%glnch,vmean,vstdev, &
           vmin, vmax)
write(ftn_stats,999) 'LWdown(W/m2)', &
                     vmean,vstdev,vmin,vmax

endif

```

```
998      FORMAT(1X,A18,4E14.3)
999      FORMAT(1X,A18,4F14.3)
```

1.36.5 noah_main.F90 (Source File: noah_main.F90)

NOAH LAND-SURFACE MODEL, UNCOUPLED 1-D COLUMN: VERSION 2.5 OCT 2001

THIS MAIN PROGRAM AND ITS FAMILY OF SUBROUTINES COMPRIZE VERSION 2.5 OF THE PUBLIC RELEASE OF THE UNCOUPLED 1-D COLUMN VERSION OF THE "NOAH" LAND-SURFACE MODEL (LSM). THE NOAH LSM IS A DESCENDANT OF AN EARLIER GENERATION OF THE OREGON STATE UNIVERSITY (OSU) LSM, BUT IT INCLUDES SUBSTANTIAL PHYSICS EXTENSIONS AND RECODING ACCOMPLISHED ALONG THE WAY BY NCEP, HL (NWS), AFGWC, AND AFGL/AFPL/AFRL. HENCE THE ACRONYM "NOAH" DENOTES N-NCEP, O-OSU, A-AIR FORCE, H-HYDRO LAB.

— FOR DOCUMENTATION OF THIS CODE AND INSTRUCTIONS ON ITS EXECUTION AND INPUT/OUTPUT FILES, SEE "NOAH LSM USERS GUIDE" IN FILE README_2.5 IN THE SAME PUBLIC SERVER DIRECTORY AS THIS SOURCE CODE.

— PROGRAM HISTORY LOG VERSION 1.0 – 01 MAR 1999

VERSION 1.1 – 08 MAR 1999

VERSION 2.0 – 27 JUL 1999

VERSION 2.1 – 23 OCT 2000

VERSION 2.2 – 07 FEB 2001

VERSION 2.3 – 07 MAY 2001 = operational Eta implementation

VERSION 2.4 – 27 JUN 2001 = ops Eta with NO physics changes

VERSION 2.5 – 18 OCT 2001

LDAS VERSION – 28 APR 2002 = NOAH Main added to LDAS Driver
(NASA GSFC)

VERSION 2.5.1– 28 MAY 2002 = Updated changes in NOAH LSM along with correction to SOILRZ and SOIL1M.

VERSION 2.6 – 24 JUN 2003 = Updated to Noah LSM v2.6

Physics changes:

in SUBROUTINE SFLX change CSOIL from 1.26E+6 to 2.00E+6

in SUBROUTINE SFLX change ZBOT from -3.0 to -8.0

Replaced de-bugged SUBROUTINE TDFCND

Removed SUBROUTINE REDPRM and moved the parameters to other locations throughout noah_main and noah_physics subroutines.

VERSION 2.5.2 – 31 MAY 2002

Fix in FUNCTION DEVAP related to FX calculation

VERSION 2.6 – Includes changes to certain parameters and snow-soil physics

INTERFACE:

```
subroutine noah_main()
```

USES:

```
use lisdrv_module, only : lis
use noah_varder      ! NOAH tile variables
use tile_spmdMod
```

CONTENTS:

```
soilmtc = 0
!==== Convert LDAS Timestep varname to NOAH timestep varname (DT) (sec)
do t = 1, di_array(iam)
  dt = lis%t%ts
!==== Bottom Temperature Field
  tbot = noah(t)%tempbot
!==== VEGETATION CLASS
```

!==== STATIC VEGETATION PARAMETERS

```
!-----  
! VEGETATION PARAMETERS ARE DEPENDENT ON VEGETATION TYPE (INDEX)  
!  
! SHDFAC: VEGETATION GREENNESS FRACTION  
!  
! RSMIN:  MINIMUM STOMATAL RESISTANCE  
!  
! RGL:    PARAMETER USED IN SOLAR RAD TERM OF  
!          CANOPY RESISTANCE FUNCTION  
!  
! HS:     PARAMETER USED IN VAPOR PRESSURE DEFICIT TERM OF  
!          CANOPY RESISTANCE FUNCTION  
!  
! SNUP:   THRESHOLD SNOW DEPTH (IN WATER EQUIVALENT M) THAT  
!          IMPLIES 100% SNOW COVER  
!  
! Z0:     Roughness Length  
!  
! XLAI:   Leaf Area Index
```

```
!-----  
! SSIB VEGETATION TYPES (DORMAN AND SELLERS, 1989; JAM)  
!  
! 1: BROADLEAF-EVERGREEN TREES (TROPICAL FOREST)  
!  
! 2: BROADLEAF-DECIDUOUS TREES  
!  
! 3: BROADLEAF AND NEEDLELEAF TREES (MIXED FOREST)  
!  
! 4: NEEDLELEAF-EVERGREEN TREES  
!  
! 5: NEEDLELEAF-DECIDUOUS TREES (LARCH)  
!  
! 6: BROADLEAF TREES WITH GROUNDCOVER (SAVANNA)  
!  
! 7: GROUNDCOVER ONLY (PERENNIAL)  
!  
! 8: BROADLEAF SHRUBS WITH PERENNIAL GROUNDCOVER  
!  
! 9: BROADLEAF SHRUBS WITH BARE SOIL  
!  
! 10: DWARF TREES AND SHRUBS WITH GROUNDCOVER (TUNDRA)  
!  
! 11: BARE SOIL  
!  
! 12: CULTIVATIONS (THE SAME PARAMETERS AS FOR TYPE 7)  
!  
! 13: GLACIAL (THE SAME PARAMETERS AS FOR TYPE 11)
```

```
NROOT = NOAH(T)%VEGP(1)
RSMIN = NOAH(T)%VEGP(2)
```

```

RGL    = NOAH(T)%VEGP(3)
HS     = NOAH(T)%VEGP(4)
SNUP   = NOAH(T)%VEGP(5)
ZO     = NOAH(T)%VEGP(6)
XLAI   = NOAH(T)%VEGP(7)

!==== MONTHLY VEGETATION PARAMETERS

SHDFAC = NOAH(T)%VEGIP

! Minimum greenness fraction
SHDMIN=0.0

! If ground surface is bare:
IF (noah(t)%VEGT .EQ. 11) SHDFAC = 0.0

!==== STATIC SOIL PARAMETERS
! -----
! SOIL PARAMETERS ARE DEPENDENT ON SOIL TYPE (INDEX)
! SMCMAX: MAX SOIL MOISTURE CONTENT (POROSITY)
! SMCREF: REFERENCE SOIL MOISTURE (ONSET OF SOIL MOISTURE
!          STRESS IN TRANSPERSION)
! SMCWLT: WILTING PT SOIL MOISTURE CONTENT
! SMCDRY: AIR DRY SOIL MOIST CONTENT LIMITS
! PSISAT: SATURATED SOIL POTENTIAL
! DKSAT: SATURATED SOIL HYDRAULIC CONDUCTIVITY
! BEXP: THE 'B' PARAMETER
! DWSAT: SATURATED SOIL DIFFUSIVITY
! F1: USED TO COMPUTE SOIL DIFFUSIVITY/CONDUCTIVITY
! QUARTZ: SOIL QUARTZ CONTENT
! -----
! SOIL TYPES ZOBLER (1986)      COSBY ET AL (1984) (quartz cont.(1))
! 1      COARSE            LOAMY SAND      (0.82)
! 2      MEDIUM           SILTY CLAY LOAM  (0.10)
! 3      FINE              LIGHT CLAY      (0.25)
! 4      COARSE-MEDIUM    SANDY LOAM      (0.60)
! 5      COARSE-FINE      SANDY CLAY      (0.52)
! 6      MEDIUM-FINE      CLAY LOAM       (0.35)
! 7      COARSE-MED-FINE  SANDY CLAY LOAM (0.60)
! 8      ORGANIC          LOAM           (0.40)
! 9      GLACIAL LAND ICE LOAMY SAND     (NA using 0.82)
! -----
NSOIL  = 4                      ! 4 soil layers in NOAH

SLDPTH(1) = 0.1                 ! Soil layer thicknesses (m)
SLDPTH(2) = 0.3
SLDPTH(3) = 0.6
SLDPTH(4) = 1.0

```

```

SOILTYP = NOAH(T)%ZOBSTOIL(1)      ! Zobler Soil Class Value

SMCMAX = NOAH(T)%SOILP(1)
PSISAT = NOAH(T)%SOILP(2)
DKSAT = NOAH(T)%SOILP(3)
BEXP = NOAH(T)%SOILP(4)
QUARTZ = NOAH(T)%SOILP(5)

! The following 5 parameters are just given here for reference
! and to force static storage allocation.

SMCREF = NOAH%SOILP(6)
SMCWLT = NOAH%SOILP(7)
SMCDRY = NOAH%SOILP(8)
DWSAT = NOAH%SOILP(9)
F1 = NOAH%SOILP(10)

!-- Here is where the above five parameters are actually derived.
! SET TWO SOIL MOISTURE WILT, SOIL MOISTURE REFERENCE PARAMETERS
SMLOW = 0.5
! Changed in 2.6 from 3 to 6 on June 2nd 2003
SMHIGH = 3.0
SMHIGH = 6.0

DWSAT = BEXP * DKSAT * (PSISAT/SMCMAX)
F1 = ALOG10(PSISAT) + BEXP*ALOG10(SMCMAX) + 2.0
SMCREF1 = SMCMAX*(5.79E-9/DKSAT)**(1.0/(2.0*BEXP+3.0))
SMCREF = SMCREF1 + (SMCMAX-SMCREF1) / SMHIGH
SMCWLT1 = SMCMAX * (200.0/PSISAT)**(-1.0/BEXP)
SMCWLT = SMCWLT1 - SMLOW * SMCWLT1
! Current version SMCDRY values equate to SMCWLT
SMCDRY = SMCWLT

! -----
! KDT IS DEFINED BY REFERENCE REFKDT AND DKSAT; REFDK=2.E-6 IS THE SAT.
! DK. VALUE FOR THE SOIL TYPE 2
! -----
REFDK=2.0E-6
REFKDT=3.0
KDT = REFKDT * DKSAT/REFDK

! -----
! FROZEN GROUND PARAMETER, FRZK, DEFINITION: ICE CONTENT THRESHOLD ABOVE
! WHICH FROZEN SOIL IS IMPERMEABLE REFERENCE VALUE OF THIS PARAMETER FOR
! THE LIGHT CLAY SOIL (TYPE=3) FRZK = 0.15 M.
! -----
FRZK=0.15

```

```

! -----
! TO ADJUST FRZK PARAMETER TO ACTUAL SOIL TYPE: FRZK * FRZFACT
! -----
FRZFACT = (SMCMAX / SMCREF) * (0.412 / 0.468)
FRZX = FRZK * FRZFACT

!==== SLOPE TYPE
! -----
! CLASS PARAMETER 'SLOPETYP' WAS INCLUDED TO ESTIMATE LINEAR RESERVOIR
! COEFFICIENT 'SLOPE' TO THE BASEFLOW RUNOFF OUT OF THE BOTTOM LAYER.
! LOWEST CLASS (SLOPETYP=0) MEANS HIGHEST SLOPE PARAMETER = 1.
! DEFINITION OF SLOPETYP FROM 'ZOBLER' SLOPE TYPE:
! SLOPE CLASS PERCENT SLOPE
! 1          0-8
! 2          8-30
! 3          > 30
! 4          0-30
! 5          0-8 & > 30
! 6          8-30 & > 30
! 7          0-8, 8-30, > 30
! 8          GLACIAL ICE
! 9          OCEAN/SEA
! ----

!-- SLOPETYP = 3
      SLOPE = 1.0

!==== MONTHLY (QUARTERLY, for now) ALBEDO (SNOW-FREE)

      ALB = NOAH(T)%ALBSF

!     Maximum Albedo over very Deep Snow

      SNOALB = NOAH(T)%MXSNALB

!==== THE FOLLOWING BREAKS DOWN THE FORCING VARIABLES
      SFCTMP = noah(t)%FORCING(1)
      Q2      = noah(t)%FORCING(2)
      SOLDN   = noah(t)%FORCING(3)
      LWDN    = noah(t)%FORCING(4)
      UWIND   = (noah(t)%FORCING(5))*(noah(t)%FORCING(5))
      VWIND   = (noah(t)%FORCING(6))*(noah(t)%FORCING(6))
      SFCSPD  = SQRT( UWIND + VWIND )
      SFCPRS  = noah(t)%FORCING(7)
      PRCP    = noah(t)%FORCING(8)
      CPCP    = noah(t)%FORCING(9)    !Convective Precipitation (kg/m2sec)

!-- Height of observations (this needs to be modified)

```

```

Z = 6.0          ! Height of observations (m)

!-- Prevent Numerical Instability for Wind Speed
  if(SFCSPD.le.0.01) SFCSPD=0.01

!-- Prevent Numerical Instability with HUMIDITY

  IF (Q2 .LT. 0.1E-5) Q2 = 0.1E-5

! Calculate Saturation Specific Humidity (Kg/Kg) and
!   Saturation vapor pressure for water (Pa) based on Specific
!   Humidity(Kg/Kg), Temperature(K), and Pressure (Pa)
!
! FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989: 'A
! SHORT COURSE IN CLOUD PHYSICS', PERGAMON PRESS, 3rd ED.
!   Pablo J. Grunmann, 3/6/98.
!
! QSAT  = Saturation Specific humidity (Kg/Kg)
! ESAT   = Saturation vapor pressure for water (Pa)
! EPS    = Water/(dry air) molecular mass ratio (epsilon)
! E      = Saturation vapor pressure
!
!-- Function E(SFCTMP) = Sat. vapor pressure (in Pascal) at
!   temperature T (uses Clausius-Clapeyron).
!
  ESAT = E(SFCTMP)

!-- CALCULATE SATURATION MIXING RATIO (PABLO GRUNMANN, 05/28/98)

  Q2SAT = 0.622 * ESAT /(SFCPRS - (1.-0.622)*ESAT)
  IF (Q2 .GE. Q2SAT) Q2 = Q2SAT*0.99

!-- CALCULATE SLOPE OF SAT. SPECIFIC HUMIDITY CURVE FOR PENMAN: DQSDT2

  DQSDT2 = DQSDT (SFCTMP, SFCPRS)

!-- CALC VIRTUAL TEMPS AND POTENTIAL TEMPS AT GRND (SUB 1) AND AT
!   THE 1ST MDL LVL ABV THE GRND (SUB 2). EXPON IS CP DIVD BY R.

  TH2 = SFCTMP + ( 0.0098 * Z )
  T2V = SFCTMP * (1.0 + 0.61 * Q2 )

  T1V = noah(t)%T1 * (1.0 + 0.61 * Q2 )
  TH2V = TH2 * (1.0 + 0.61 * Q2 )

!== OPTIONAL SUBROUTINE: Calculate LW Radiation (Down) =====
!   CALL OBTLWDN(SFCTMP,LWDN)

```

```
!-- Photo Thermal Unit (PTU)

PTU      = 0.10

!-- Initialize SOILM for 1st timestep water balance
!     SOILM = 0.0
!-- Initialize ROOT ZONE COLUMN SOIL MOISTURE IN METERS (SOILRZ)
!     SOILRZ = 0.0
!-- Initialize TOP 1-METER COLUMN SOIL MOISTURE IN METERS (SOIL1M)
!     SOIL1M = 0.0

!= CALCULATE CH (EXCHANGE COEFFICIENT) =====
!
!     CH IS THE SFC EXCHANGE COEFFICIENT FOR HEAT/MOISTURE
!     CM IS THE SFC EXCHANGE COEFFICIENT FOR MOMENTUM
!
! IMPORTANT NOTE: TO CALCULATE THE SFC EXCHANGE COEF (CH) FOR HEAT AND
!                 MOISTURE, SUBROUTINE SFCDIF BELOW CAN:
!
!     A) BE CALLED HERE FROM THE DRIVER, THUS CH IS INPUT TO SFLX
!        (AS IS TYPICAL IN A COUPLED ATMOSPHERE/LAND MODEL), OR
! * B) BE CALLED INTERNALLY IN ROUTINE SFLX (THUS CH IS OUTPUT FROM SFLX),
!        BEFORE THE CALL TO ROUTINE "PENMAN"
!
! OPTION B IS THE DEFAULT HERE. THAT IS, IN THE UNCOUPLED, OFF-LINE LSM
! REPRESENTED HEREIN BY THIS DRIVER, WE CALL SFCDIF LATER IN ROUTINE SFLX.
!
! THE ROUTINE SFCDIF REPRESENTS THE SO-CALLED "SURFACE LAYER" OR THE
! "CONSTANT FLUX LAYER" (THE LOWEST 20-100 M OF THE ATMOSPHERE).
! HENCE ROUTINE SFCDIF EMBODIES THE "ATMOSPHERIC AERODYNAMIC RESISTANCE".
!
! TO ENABLE THE FLEXIBILITY OF EITHER OPTION A OR B, WE PASS
! THE ARGUMENTS "CH", "CM", AND "SFCSPD" (WIND SPEED:JUST CALCULATED ABOVE)
! TO ROUTINE SFLX TO SUPPORT OPTION B -- THAT IS, FOR INPUT TO THE CALL TO
! ROUTINE SFCDIF THEREIN. IN OPTION A, THE ARGUMENTS "SFCSPD" AND "CM"
! ARE NEITHER NEEDED IN ROUTINE SFLX, NOR ALTERED BY ROUTINE SFLX.
!
! IF ONE CHOOSES OPTION A, THEN ONE MUST
!   1 - ACTIVATE (UNCOMMENT) THE CALL TO SFCDIF BELOW,
!   2 - ACTIVATE (UNCOMMENT) THE ASSIGNMENT OF "ZO" AND "CZIL" NEXT BELOW
!   3 - DE-ACTIVATE (COMMENT OUT) THE CALL TO SFCDIF IN ROUTINE SFLX.
!
! ZO and CZIL:
!
! THE ROUGHNESS LENGTH PARAMETERS "ZO" AND "CZIL" MUST BE SET HERE IN THE
! DRIVER TO SUPPORT THE "OPTION-A", I.E. THE CALL TO SFCDIF BELOW. IN SO
! DOING, THE "ZO" AND "CZIL" ASSIGNED HERE MUST CORRESPOND TO THEIR
! VALUES, CALLED BY SFLX JUST BEFORE CALL SFCDIF.
```

```

!  THUS THE VALUE OF "ZO" ASSIGNED HERE MUST CORRESPOND TO THAT ASSIGNED
!  FOR THE CHOSEN VEG CLASS THAT WAS ALREADY INPUT EARLIER IN THE DRIVER.
!

!  BECAUSE OF THE IMPLICIT ITERATIVE NATURE OF THE "PAULSON" SURFACE-LAYER
!  SCHEME USED IN ROUTINE SFCDIF, CH AND CM ARE CO-DEPENDENT.  SIMILARLY,
!  THE IMPLICIT NATURE OF THE SFCDIF SCHEME ALSO REQUIRES THAT FOR EITHER
!  OPTION A OR B, CH AND CM MUST BE INITIALIZED EARLIER IN THE DRIVER BEFORE
!  THE START OF THE TIME-STEP LOOP, AS WELL AS BE CARRIED FORWARD FROM
!  TIME STEP TO TIME STEP AS "STATE VARIABLES", BECAUSE THE VALUES OF
!  CH AND CM FROM A PREVIOUS TIME STEP REPRESENT THE FIRST-GUESS VALUES FOR
!  THE CALL TO SFCDIF IN THE PRESENT TIME STEP.

!  SOME USERS MAY CHOOSE TO EXECUTE AN ENTIRELY DIFFERENT SCHEME IN PLACE OF
!  ROUTINE SFCDIF HERE, E.G. AN EXPLICIT SCHEME SUCH AS LOUIS (1979) THAT
!  EMPLOYS NO ITERATION AND HAS NO REQUIREMENT TO CARRY CH AND CM FORWARD
!  AS STATE VARIABLES FROM TIME STEP TO TIME STEP.  IN THAT CASE, IN
!  OPTION A, THE ROUTINE SHOULD BE CALLED HERE IN THE DRIVER AFTER ALL
!  NECESSARY INPUT ARGUMENTS FOR IT ARE DEFINED AT THIS POINT, OR CALLED IN
!  ROUTINE SFLX, AT THE POINT SFCDIF IS CALLED.

!  CALL SFCDIF ( Z, ZO, T1V, TH2V, SFCSPD,CZIL, CM, CH )

! -----
!  INITIALIZE CH, CM (NOTE: initial these before time loop)
!    CH=1.E-4
!    CM=1.E-4
!  1998 May 22 0030 (Julian= 142) typical values initialization
!    CH= 0.0150022404
!    CM= 0.0205970779
!  **NOTE: TRYING THESE VALUES AS TEST!
! -----|
```

==== MAIN CALL TO LAND-SURFACE PHYSICS <<<<<<<<<<<<<<<<

```

CALL SFLX (T,&
ICE,DT,Z,NSOIL,SLDPTH,&
LWDN,SOLDN,SFCPRS,PRCP,SFCTMP,Q2,SFCSPD, &
TH2,Q2SAT,DQSDT2,&
SLOPE,SHDFAC,SHDMIN,PTU,ALB,SNOALB, &
RSMIN,RGL,HS,SNUP,ZO,XLAI,NROOT,&
PSISAT,BEXP,DKSAT,SMCMAX,QUARTZ,DWSAT, &
SMCWLT,SMCREF,SMCDRY,F1,KDT,FRZX,FRZFACT,TBOT, &
NOAH(T)%CMC,NOAH(T)%T1,NOAH(T)%STC,NOAH(T)%SMC,NOAH(T)%SH20, &
NOAH(T)%SNOWH,NOAH(T)%SNEQV,ALBEDO,NOAH(T)%CH,NOAH(T)%CM,&
EVP,ETA,SHTFLX, &
EC,EDIR,ET,ETT,ESNOW,DRIP,DEW, &
BETA,ETP,GFLX, &
FLX1,FLX2,FLX3,&
SNOMLT,SNCOVR,&
```

```

RUNOFF1,RUNOFF2,RUNOFF3, &
RC,PC,RCS,RCT,RCQ,RCSOIL, &
MSTAVRZ,MSTAVTOT,SOILM)
!***** ****
!
!    CALCULATE UPWARD LONGWAVE RAD USING UPDATED SKIN TEMPERATURE
!
T14 = noah(t)%T1 * noah(t)%T1 * noah(t)%T1 * noah(t)%T1
FUP = 5.67E-8 * T14
!
!    CALCULATE RESIDUAL OF ALL SURFACE ENERGY BALANCE EQN TERMS.
!
!    GFLX = -GFLX
!    F = SOLDN*(1.0-ALBEDO) + LWDN
!    RES      = F - SHTFLX - GFLX - ETA - FUP - FLX1 - FLX2 - FLX3
!    ENDIF
!
!    PRINT*, ' -----',
!    PRINT*, ' State Variables ',
!    PRINT*, ' -----',
!    WRITE(*,*) NOAH(T)%T1,' T1...Skin temperature (K)'
!    WRITE(*,*)(NOAH(T)%STC(IJ), IJ=1,NSOIL),' STC'
!    WRITE(*,*)(NOAH(T)%SMC(IJ), IJ=1,NSOIL),' SMC'
!    WRITE(*,*)(NOAH(T)%SH20(IJ), IJ=1,NSOIL),' SH20'
!    WRITE(*,*) NOAH(T)%CMC,' CMC...Canopy water content (m)'
!    WRITE(*,*) NOAH(T)%SNOWH,' SNOWH...Actual snow depth (m)'
!    WRITE(*,*) NOAH(T)%SNEQV,' SNEQV...Water equiv snow depth (m)'
!    WRITE(*,*) 'CH= ',NOAH(T)%CH,' CM= ',NOAH(T)%CM
!    PRINT*, ' -----',
!
!== Collect the output variables into NOAH(T)%RETURN
noah(t)%swnet = noah(t)%swnet+soldn*(1.0-albedo)
noah(t)%lwnet = noah(t)%lwnet+(5.67E-8)*(NOAH(T)%T1**4.0)-LWDN
noah(t)%qle = noah(t)%qle+eta
noah(t)%qh = noah(t)%qh+shtflx
noah(t)%qg = noah(t)%qg-gflx
!
if (sfctmp .le. t0) then
    noah(t)%snowf = noah(t)%snowf+prcp
    noah(t)%rainf = noah(t)%rainf+0.0
else
    noah(t)%snowf = noah(t)%snowf+0.0
    noah(t)%rainf = noah(t)%rainf+prcp
endif
noah(t)%evap = noah(t)%evap+evp
noah(t)%qs = noah(t)%qs+runoff1*1000.0
noah(t)%qsb = noah(t)%qsb+ runoff2*1000.0

```

```

noah(t)%qsm = noah(t)%qsm+ snomlt*1000.0
NOAH(T)%avgsurft=NOAH(T)%T1
NOAH(T)%albedo=ALBEDO
noah(t)%swe = noah(t)%swe + noah(t)%sneqv*1000.0

NOAH(T)%soilmoist1= noah(t)%soilmoist1+NOAH(T)%SMC(1)*1000.0*SLDPTH(1)
NOAH(T)%soilmoist2 =noah(t)%soilmoist2+NOAH(T)%SMC(2)*1000.0*SLDPTH(2)
NOAH(T)%soilmoist3 =noah(t)%soilmoist3+NOAH(T)%SMC(3)*1000.0*SLDPTH(3)
NOAH(T)%soilmoist4 =noah(t)%soilmoist4+NOAH(T)%SMC(4)*1000.0*SLDPTH(4)

NOAH(T)%soilwet =noah(t)%soilwet+MSTAVTOT

NOAH(T)%tveg = noah(t)%tveg+ETT*1000.0
NOAH(T)%esoil = noah(t)%esoil+EDIR*1000.0
! ROOT ZONE COLUMN SOIL MOISTURE IN METERS (SOILRZ)
do k = 1,nroot
    soilrz = soilrz+(noah(t)%smc(k)*sldpth(k)*1000.0)
end do
noah(t)%rootmoist= noah(t)%rootmoist+soilrz

if(lis%t%tscount == 0 .or. lis%t%tscount ==1) then

    soilmtc(t) = soilmtc(t)+noah(t)%soilmoist1+ &
        noah(t)%soilmoist2+ noah(t)%soilmoist3+&
        noah(t)%soilmoist4
    noah(t)%soilm_prev = soilmtc(t)
    noah(t)%swe_prev = noah(t)%swe
endif

noah(t)%count=noah(t)%count+1

enddo
return

///////////////////////////////
{\sf CONTENTS:}
\begin{verbatim}
end subroutine noah_main

*** NOAH FUNCTIONS ****
FUNCTION DQS (T)

IMPLICIT NONE

!
! PURPOSE: TO CALCULATE VALUES OF VAPOR PRESSURE (E)

```

! AND P * DQS/DT (P TIMES CHG IN SAT MXG RATIO WITH RESPECT
! TO THE CHG IN TEMP) IN SUBSTITUTION TO THE LOOK-UP TABLES.

!
! FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989.
! ADDED BY PABLO J. GRUNMANN, 6/30/97.
!

REAL DESDT

REAL DQS

REAL ESD

REAL LW

REAL T

REAL ES

! REAL, PARAMETER:: CP = 1005.

! REAL, PARAMETER:: CV = 718.

! REAL, PARAMETER:: CVV = 1410.

REAL, PARAMETER:: CPV = 1870.

REAL, PARAMETER:: RV = 461.5

REAL, PARAMETER:: CW = 4187.

REAL, PARAMETER:: EPS = 0.622

REAL, PARAMETER:: ESO = 611.2

REAL, PARAMETER:: TO = 273.15

REAL, PARAMETER:: LVH2O = 2.501000E+6

! ABOUT THE PARAMETERS:

! EPS ----- WATER - DRY AIR MOLECULAR MASS RATIO, EPSILON

! VALUES FOR SPECIFIC HEAT CAPACITY AND INDIVIDUAL GAS CONSTANTS
! IN [JOULES/(KG*KELVIN)] UNITS.

! DRY AIR:

CP, CV

WATER VAPOR:

CVV = 1410.

CPV = 1870.

RV = 461.5

LIQUID WATER:

CW = 4187.

! ESO = ES(T=273.15 K) = SAT. VAPOR PRESSURE (IN PASCAL) AT T=TO
! TO = 273.15

! SAT. MIXING RATIO: QS ~= EPS*ES/P

! CLAUSIUS-CLAPEYRON: DES/DT = L*ES/(RV*T^2)

! @QS/@T = (EPS/P)*DES/DT

```

LW = LVH20 - ( CW - CPV ) * ( T - TO )
ES = ESO*EXP (LW*(1/TO - 1/T)/RV)
DESDT = LW*ES/(RV*T*T)

! FOR INSERTION IN DQSDT FUNCTION:
! DQSDT = DQS/P , WHERE DQS = EPS*DESDT

DQS = EPS*DESDT

RETURN
END

!-----
FUNCTION DQSDT ( SFCTMP, SFCPRS )

IMPLICIT NONE

!
! PURPOSE: TO RETRIEVE THE APPROPRIATE VALUE OF DQSDT (THE CHANGE
! ====== OF THE SATURATION MIXING RATIO WITH RESPECT TO THE
! CHANGE IN TEMPERATURE) FROM:
!
! FORMULAS INTRODUCED IN FUNCTION DQS
! (MODIFIED BY PABLO GRUNMANN, 7/9/97).
!

REAL SFCTMP
REAL SFCPRS
REAL DQS
REAL DQSDT

IF ((SFCTMP .GE. 173.0) .AND. (SFCTMP .LE. 373.0)) THEN

! IF THE INPUT SFC AIR TEMP IS BTWN 173 K AND 373 K, USE
! FUNCTION DQS TO DETERMINE THE SLOPE OF SAT.MIX RATIO FUNCTION

DQSDT = DQS (SFCTMP) / SFCPRS

ELSE

! OTHERWISE, SET DQSDT EQUAL TO ZERO

DQSDT = 0.0

END IF

```

```
RETURN
END

!-----
FUNCTION E(T)

IMPLICIT NONE

!
! PURPOSE: TO CALCULATE VALUES OF SAT. VAPOR PRESSURE (E)
! FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989.
! ADDED BY PABLO J. GRUNMANN, 7/9/97.
!

REAL LW
REAL T
REAL E

! REAL, PARAMETER:: EPS = 0.622
! REAL, PARAMETER:: CP = 1005.
! REAL, PARAMETER:: CV = 718.
! REAL, PARAMETER:: CVV = 1410.
! REAL, PARAMETER:: CPV = 1870.
! REAL, PARAMETER:: RV = 461.5
! REAL, PARAMETER:: CW = 4187.
! REAL, PARAMETER:: ESO = 611.2
! REAL, PARAMETER:: TO = 273.15
! REAL, PARAMETER:: LVH2O = 2.501000E+6

! ABOUT THE PARAMETERS:

! EPS --- WATER - DRY AIR MOLECULAR MASS RATIO, EPSILON

! VALUES FOR SPECIFIC HEAT CAPACITY AND INDIVIDUAL GAS CONSTANTS
! IN [JOULES/(KG*KELVIN)] UNITS.

! DRY AIR:
! CP, CV
! WATER VAPOR:
! CVV = 1410.
! CPV = 1870.
! RV = 461.5
! LIQUID WATER:
! CW = 4187.

! ESO = ES(TO) = SAT. VAPOR PRESSURE (IN PASCAL) AT T=TO
! TO = 273.15
```

```
!  
!     CLAUSIUS-CLAPEYRON: DES/DT = L*ES/(RV*T^2)  
  
    LW = LVH20 - ( CW - CPV ) * ( T - TO )  
    E = ESO*EXP (LW*(1/TO - 1/T)/RV)  
  
    RETURN  
    END  
  
!  
!CC 1. DRIVER SUBROUTINE ==> SUBROUTINE OBTLWDN CCCCCCCCCCCCCCCCCC  
  
!     SUBROUTINE OBTLWDN(SFCTMP,LWDN)  
  
!             RADIATION  
  
!     The following step (OBTENTION OF LWDN) is used if  
!     user wants to calculate longwave downward.  
!  
!     OBTENTION OF LWDN <<<<<<<<<<<<<<<<<<<<<<<  
!  
!     COMPUTATION OF LWDN (INCOMING LW RADIATION) FROM TAIR AND Q:  
!  
!.....LWDN = EMISS*SIGMA*(TAK)^4.  
!  
! WHERE:   TAK = AIR TEMP IN KELVIN  
!          EMISS = 0.7 OR (IDS0 AND JACKSON, 1969):  
!  
!          EMISS = (1 - 0.261 EXP(-7.77*10^(-4)*(273-TAK)^2))  
!  
! NEED STEFAN-BOLTZMANN CONSTANT, SIGMA  
!          SIGMA = 5.672 * 10^-8 W M^-2 T^-4  
!  
!          SIGMA = 5.672E-8  
!          TAK = SFCTMP  
!          EMISS = 1 - 0.261*EXP((-7.77E-4)*(273-TAK)^2.)  
!  
!          LWDN = EMISS*SIGMA*TAK^4.  
!  
!          RETURN  
!          END  
  
!CCCC END OF DRIVER SUBROUTINES CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

\markboth{Left}{Source File: noah_module.F90, Date: Mon Dec 22 15:57:49 EST 2003}
}

!---

```
!      NASA GSFC Land Information Systems LIS 2.3      !
!-----
```

1.37 Fortran: Module Interface noah_module.F90 (Source File: noah_module.F90)

Module for 1-D NOAH land model driver variable specification.

REVISION HISTORY:

28 Apr 2002: K. Arsenault added NOAH LSM 2.5 code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS

INTERFACE:

```
module noah_module
```

CONTENTS:

```
type noahdec

    INTEGER :: ts                      !Timestep (seconds)
    INTEGER :: maxt                     !Maximum tiles per grid
    INTEGER :: SIBVEG                  !UMD to SIB Vegetation Class Index value
    INTEGER :: NSLAY                    !Number of NOAH soil layers (4)
    INTEGER :: COUNT
    INTEGER :: ZOBSOIL(1)   !Zobler Soil Classes (LIS%NCH)

    REAL:: VEGP(7)        !Static vegetation parameter values, dim(NOAH_NVEGP)
    REAL:: VEGIP          !Interpolated Green Fraction from monthly parameters
    REAL:: VEGMP1         !Month 1 Greenness Fraction Value
    REAL:: VEGMP2         !Month 2 Greenness Fraction Value
    REAL:: ALBSF1         !Date 1 Snow-Free Albedo Value
    REAL:: ALBSF2         !Date 2 Snow-Free Albedo Value
    REAL:: SOILP(10)       !Static soil parameter values, dim(NOAH_NSOILP)
    REAL:: ALBSF          !Quarterly Snow-Free Albedo dataset
    REAL:: MXSNALB        !Maximum snow albedo dataset
    REAL:: TEMPBOT        !Bottom boundary temperature
!-----
! NOAH-State Variables
!-----
    REAL :: T1                      !NOAH Skin Temperature (K)
    REAL :: CMC                     !NOAH Canopy Water Content
    REAL :: SNOWH                   !NOAH Actual Snow depth (m)
    REAL :: SNEQV                   !NOAH Water Equivalent Snow Depth (m)
    REAL :: STC(4)                 !NOAH Soil Temperaure (4 layers)
    REAL :: SMC(4)                 !NOAH Soil (4 layers)
    REAL :: SH2O(4)                !NOAH Liquid-only soil moisture (4 layers)
```

```

REAL :: CH           !NOAH Heat/moisture exchange coef.
REAL :: CM           !NOAH Momentum exchange coef.
REAL :: FORCING(10)   ! TILE FORCING..
REAL :: VEGT          !vegetation type of tile
!-----
!  

! NOAH-Output variables
!-----
REAL :: swnet
REAL :: lwnet
REAL :: qle
REAL :: qh
REAL :: qg
REAL :: snowf
REAL :: rainf
REAL :: evap
REAL :: qs
REAL :: qsb
REAL :: qsm
REAL :: avgsurft
REAL :: albedo
REAL :: swe
REAL :: soilmoist1
REAL :: soilmoist2
REAL :: soilmoist3
REAL :: soilmoist4
REAL :: soilwet
REAL :: tveg
REAL :: esoil
REAL :: rootmoist
REAL :: soilm_prev
REAL :: swe_prev
end type noahdec

```

1.37.1 noah_output.F90 (Source File: noah_output.F90)

This subroutines sets up methods to write noah output

INTERFACE:

```
subroutine noah_output
```

USES:

```

use lisdrv_module, only : lis, tile, glbgindex
use noah_varder, only : noahdrv
use spmdMod, only : masterproc,npes

```

CONTENTS:

```

    if(lis%o%wsingle ==1) then
!-----
! Writes each output variable to a separate file
!-----
    if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
        do i=1,32
            call noah_singlegather(i,var)
            if(masterproc) then
                call noah_singleout(lis, tile, glbgindex, var, i)
            endif
        enddo
        call noah_totinit()
    endif
else
!-----
! Writes bundled output
!-----
    if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
        if(npes > 1 ) then
            call noah_gather()
        endif
        if(masterproc) then
            call noah_almaout(lis, tile, glbgindex)
        endif
        call noah_totinit()
    endif
endif

```

1.38 Fortran: Module Interface noahpardef_module.F90 (Source File: noahpardef_module.F90)

This module contains routines that defines MPI derived data types for Noah LSM

REVISION HISTORY:

06 Oct 2003; Sujay Kumar Initial Specification

INTERFACE:

```
module noahpardef_module
```

USES:

```

use noah_module
use noahdrv_module
use spmdMod
implicit none
```

ARGUMENTS:

```
integer:: MPI_NOAH_STRUCT !MPI derived type for noah$-$module
integer :: MPI_NOAHDVR_STRUCT !MPI derived type for noahdrv$-$module
```

1.38.1 def_noahpar_struct (Source File: noahpardef_module.F90)

Routine that defines MPI derived data types for Noah

INTERFACE:

```
subroutine def_noahpar_struct()
```

!ROUTINE : noah_physics.F90

DESCRIPTION: SUB-DRIVER FOR "NOAH/OSU LSM" FAMILY OF PHYSICS SUBROUTINES FOR A SOIL/VEG/SNOWPACK LAND-SURFACE MODEL TO UPDATE SOIL MOISTURE, SOIL ICE, SOIL TEMPERATURE, SKIN TEMPERATURE, SNOWPACK WATER CONTENT, SNOWDEPTH, AND ALL TERMS OF THE SURFACE ENERGY BALANCE AND SURFACE WATER BALANCE (EXCLUDING INPUT ATMOSPHERIC FORCINGS OF DOWNWARD RADIATION AND PRECIP)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault; Added NOAH LSM 2.5 code to LDAS
15 May 2002: Urszula Jambor; Changed LOGICAL to LOGICAL*1 to match new
              GRIB libraries
28 May 2002: Kristi Arsenault; Updated NOAH code to 2.5.1, and
              corrected problem with FX in DEVAP function.
12 Jun 2002: Kristi Arsenault; Updated NOAH code to 2.5.2
04 Nov 2002: Kristi Arsenault; Incorporated new TBOT fields
24 Jun 2003: Kristi Arsenault; Updated Noah LSM to 2.6 version
```

----- THE FOLLOWING SUBROUTINES ARE IN ALPHABETICAL ORDER -----

-- 1. PHYSICS SUBROUTINE ==> SUBROUTINE ALCALC -----

SUBROUTINE ALCALC (ALB,SNOALB,SHDFAC,SHDMIN,SNCOVR,TSNOW,ALBEDO)

IMPLICIT NONE

CALCULATE ALBEDO INCLUDING SNOW EFFECT (0 -> 1)

ALB	SNOWFREE ALBEDO
SNOALB	MAXIMUM (DEEP) SNOW ALBEDO
SHDFAC	AREAL FRACTIONAL COVERAGE OF GREEN VEGETATION
SHDMIN	MINIMUM AREAL FRACTIONAL COVERAGE OF GREEN VEGETATION
SNCOVR	FRACTIONAL SNOW COVER

```
ALBEDO SURFACE ALBEDO INCLUDING SNOW EFFECT
TSNOW SNOW SURFACE TEMPERATURE (K)
```

```
REAL ALB, SNOALB, SHDFAC, SHDMIN, SNCovR, ALBEDO, TSnow
```

```
SNOALB IS ARGUMENT REPRESENTING MAXIMUM ALBEDO OVER DEEP SNOW,
AS PASSED INTO SFLX, AND ADAPTED FROM THE SATELLITE-BASED MAXIMUM
SNOW ALBEDO FIELDS PROVIDED BY D. ROBINSON AND G. KUKLA
(1985, JCAM, VOL 24, 402-411)
```

```
changed in version 2.6 on June 2nd 2003
```

```
ALBEDO = ALB + (1.0-(SHDFAC-SHDMIN))*SNCovR*(SNOALB-ALB)
ALBEDO = ALB + SNCovR*(SNOALB-ALB)
IF (ALBEDO .GT. SNOALB) ALBEDO=SNOALB
```

```
BASE FORMULATION (DICKINSON ET AL., 1986, COGLEY ET AL., 1990)
IF (TSnow.LE.263.16) THEN
    ALBEDO=SNOALB
ELSE
    IF (TSnow.LT.273.16) THEN
        TM=0.1*(TSnow-263.16)
        ALBEDO=0.5*((0.9-0.2*(TM**3))+(0.8-0.16*(TM**3)))
    ELSE
        ALBEDO=0.67
    ENDIF
ENDIF
```

```
ISBA FORMULATION (VERSEGHEY, 1991; BAKER ET AL., 1990)
IF (TSnow.LT.273.16) THEN
    ALBEDO=SNOALB-0.008*DT/86400
ELSE
    ALBEDO=(SNOALB-0.5)*EXP(-0.24*DT/86400)+0.5
ENDIF
```

```
END SUBROUTINE ALCALC
```

```
RETURN
END
```

CCC 2. PHYSICS SUBROUTINE ==> SUBROUTINE CANRES CCCCCCCCCCCCCCCCCCC

```
SUBROUTINE CANRES (SOLAR,CH,SFCTMP,Q2,SFCPRS,SMC,ZSOIL,NSOIL, &
SMCWLT,SMCREF,RSMIN,RC,PC,NROOT,Q2SAT,DQSDT2, &
TOPT,RSMAX,RGL,HS,XLAI, &
RCS,RCT,RCQ,RCSOIL)
```

IMPLICIT NONE

SUBROUTINE CANRES

CALCULATE CANOPY RESISTANCE WHICH DEPENDS ON INCOMING SOLAR RADIATION,
AIR TEMPERATURE, ATMOSPHERIC WATER VAPOR PRESSURE DEFICIT AT THE
LOWEST MODEL LEVEL, AND SOIL MOISTURE (PREFERABLY UNFROZEN SOIL
MOISTURE RATHER THAN TOTAL)

SOURCE: JARVIS (1976), NOILHAN AND PLANTON (1989, MWR), JACQUEMIN AND
NOILHAN (1990, BLM)

SEE ALSO: CHEN ET AL (1996, JGR, VOL 101(D3), 7251-7268), EQNS 12-14
AND TABLE 2 OF SEC. 3.1.2

INPUT:

SOLAR INCOMING SOLAR RADIATION
 CH SURFACE EXCHANGE COEFFICIENT FOR HEAT AND MOISTURE
 SFCTMP AIR TEMPERATURE AT 1ST LEVEL ABOVE GROUND
 Q2 AIR HUMIDITY AT 1ST LEVEL ABOVE GROUND
 Q2SAT SATURATION AIR HUMIDITY AT 1ST LEVEL ABOVE GROUND
 DQSDT2 SLOPE OF SATURATION HUMIDITY FUNCTION WRT TEMP
 SFCPRS SURFACE PRESSURE
 SMC VOLUMETRIC SOIL MOISTURE
 ZSOIL SOIL DEPTH (NEGATIVE SIGN, AS IT IS BELOW GROUND)
 NSOIL NO. OF SOIL LAYERS
 NROOT NO. OF SOIL LAYERS IN ROOT ZONE (1.LE.NROOT.LE.NSOIL)
 XLAI LEAF AREA INDEX
 SMCWLT WILTING POINT
 SMCREF REFERENCE SOIL MOISTURE (WHERE SOIL WATER DEFICIT STRESS
 SETS IN)

RSMIN, RSMAX, TOPT, RGL, HS ARE CANOPY STRESS PARAMETERS SET IN
 SUBROUTINE REDPRM

OUTPUT:

PC PLANT COEFFICIENT
 RC CANOPY RESISTANCE

INTEGER NSOLD
 PARAMETER(NSOLD = 20)

INTEGER K
 INTEGER NROOT
 INTEGER NSOIL

REAL CH
 REAL CP
 REAL DELTA
 REAL DQSDT2

```
REAL FF
REAL GX
REAL HS
REAL P
REAL PART(NSOLD)
REAL PC
REAL Q2
REAL Q2SAT
REAL RC
REAL RSMIN
REAL RCQ
REAL RCS
REAL RCSOIL
REAL RCT
REAL RD
REAL RGL
REAL RR
REAL RSMAX
REAL SFCPRS
REAL SFCTMP
REAL SIGMA
REAL SLV
REAL SMC(NSOIL)
REAL SMCREF
REAL SMCWLT
REAL SOLAR
REAL TOPT
REAL SLVCP
REAL ST1
REAL TAIR4
REAL XLAI
REAL ZSOIL(NSOIL)

PARAMETER(CP = 1004.5)
PARAMETER(RD = 287.04)
PARAMETER(SIGMA = 5.67E-8)
PARAMETER(SLV = 2.501000E6)
```

```
INITIALIZE CANOPY RESISTANCE MULTIPLIER TERMS.
```

```
RCS = 0.0
RCT = 0.0
RCQ = 0.0
RCSOIL = 0.0
RC = 0.0
```

CONTRIBUTION DUE TO INCOMING SOLAR RADIATION

```
FF = 0.55*2.0*SOLAR/(RGL*XLA)  
RCS = (FF + RSMIN/RSMAX) / (1.0 + FF)  
RCS = MAX(RCS,0.0001)
```

CONTRIBUTION DUE TO AIR TEMPERATURE AT FIRST MODEL LEVEL ABOVE GROUND
RCT EXPRESSION FROM NOILHAN AND PLANTON (1989, MWR).

```
RCT = 1.0 - 0.0016*((TOPT-SFCTMP)**2.0)  
RCT = MAX(RCT,0.0001)
```

CONTRIBUTION DUE TO VAPOR PRESSURE DEFICIT AT FIRST MODEL LEVEL.
RCQ EXPRESSION FROM SSIB

```
RCQ = 1.0/(1.0+HS*(Q2SAT-Q2))  
RCQ = MAX(RCQ,0.01)
```

CONTRIBUTION DUE TO SOIL MOISTURE AVAILABILITY.
DETERMINE CONTRIBUTION FROM EACH SOIL LAYER, THEN ADD THEM UP.

```
GX = (SMC(1) - SMCWLT) / (SMCREF - SMCWLT)  
IF (GX .GT. 1.) GX = 1.  
IF (GX .LT. 0.) GX = 0.
```

USE SOIL DEPTH AS WEIGHTING FACTOR

```
PART(1) = (ZSOIL(1)/ZSOIL(NROOT)) * GX
```

USE ROOT DISTRIBUTION AS WEIGHTING FACTOR

```
PART(1) = RTDIS(1) * GX
```

```
DO K = 2,NROOT  
GX = (SMC(K) - SMCWLT) / (SMCREF - SMCWLT)  
IF (GX .GT. 1.) GX = 1.  
IF (GX .LT. 0.) GX = 0.
```

USE SOIL DEPTH AS WEIGHTING FACTOR

```
PART(K) = ((ZSOIL(K)-ZSOIL(K-1))/ZSOIL(NROOT)) * GX
```

USE ROOT DISTRIBUTION AS WEIGHTING FACTOR

```
PART(K) = RTDIS(K) * GX
```

```

END DO

DO K = 1,NROOT
    RCSOIL = RCSOIL+PART(K)
END DO
RCSOIL = MAX(RCSOIL,0.0001)

```

DETERMINE CANOPY RESISTANCE DUE TO ALL FACTORS. CONVERT CANOPY RESISTANCE (RC) TO PLANT COEFFICIENT (PC) TO BE USED WITH POTENTIAL EVAP IN DETERMINING ACTUAL EVAP. PC IS DETERMINED BY:

PC * LINERIZED PENMAN POTENTIAL EVAP =
PENMAN-MONTEITH ACTUAL EVAPORATION (CONTAINING RC TERM).

```
RC = RSMIN/(XLAI*RCS*RCT*RCQ*RCSOIL)
```

```

TAIR4 = SFCTMP**4.
ST1 = (4.*SIGMA*RD)/CP
SLVCP = SLV/CP
RR = ST1*TAIR4/(SFCPRS*CH) + 1.0
RR = (4.*SIGMA*RD/CP)*(SFCTMP**4.)/(SFCPRS*CH) + 1.0
DELTA = (SLV/CP)*DQSRT2

PC = (RR+DELTA)/(RR*(1.+RC*CH)+DELTA)

```

END SUBROUTINE CANRES

```

RETURN
END

```

FUNCTION CSNOW (DSNOW)

IMPLICIT NONE

FUNCTION CSNOW

CALCULATE SNOW THERMAL CONDUCTIVITY

```

REAL C
REAL DSNOW
REAL CSNOW
REAL UNIT

```

```
PARAMETER(UNIT = 0.11631)
```

```
-----  
CSNOW IN UNITS OF CAL/(CM*HR*C) , RETURNED IN W/(M*C)  
BASIC VERSION IS DYACHKOVA EQUATION (1960), FOR RANGE 0.1-0.4  
-----
```

```
C=0.328*10**2.25*DSNOW)  
CSNOW=UNIT*C
```

```
-----  
DE VAUX EQUATION (1933), IN RANGE 0.1-0.6  
-----
```

```
CSNOW=0.0293*(1.+100.*DSNOW**2)
```

```
-----  
E. ANDERSEN FROM FLERCHINGER  
-----
```

```
CSNOW=0.021+2.51*DSNOW**2
```

```
-----  
END FUNCTION CSNOW  
-----
```

```
RETURN  
END
```

```
FUNCTION DEVAP (ETP1,SMC,ZSOIL,SHDFAC,SMCMAX,BEXP, &  
DKSAT,DWSAT,SMCDRY,SMCREF,SMCWLT,FXEXP)
```

```
IMPLICIT NONE
```

```
-----  
FUNCTION DEVAP  
-----
```

```
CALCULATE DIRECT SOIL EVAPORATION  
-----
```

```
REAL BEXP  
REAL DEVAP  
REAL DKSAT  
REAL DWSAT  
REAL ETP1  
REAL FX  
REAL FXEXP  
REAL SHDFAC  
REAL SMC  
REAL SMCDRY  
REAL SMCMAX  
REAL ZSOIL  
REAL SMCREF  
REAL SMCWLT
```

DIRECT EVAP A FUNCTION OF RELATIVE SOIL MOISTURE AVAILABILITY, LINEAR
WHEN FXEXP=1.

FX = ((SMC - SMCDRY) / (SMCMAX - SMCDRY))**FXEXP

FX > 1 REPRESENTS DEMAND CONTROL
FX < 1 REPRESENTS FLUX CONTROL

FX = MAX (MIN (FX, 1.), 0.)

ALLOW FOR THE DIRECT-EVAP-REDUCING EFFECT OF SHADE

DEVAP = FX * (1.0 - SHDFAC) * ETP1

END FUNCTION DEVAP

RETURN
END

CCCC 3. PHYSICS SUBROUTINE ==> SUBROUTINE HRT CCCCCCCCCCCCCCCCCCCCC

SUBROUTINE HRT (RHSTS,STC,SMC,SMCMAX,NSOIL,ZSOIL,YY,ZZ1, &
TBOT,ZBOT,PSISAT,SH20,DT,BEXP, &
F1,DF1,QUARTZ,CSOIL,AI,BI,CI)

IMPLICIT NONE

SUBROUTINE HRT

CALCULATE THE RIGHT HAND SIDE OF THE TIME TENDENCY TERM OF THE SOIL
THERMAL DIFFUSION EQUATION. ALSO TO COMPUTE (PREPARE) THE MATRIX
COEFFICIENTS FOR THE TRI-DIAGONAL MATRIX OF THE IMPLICIT TIME SCHEME.

INTEGER NSOLD
PARAMETER(NSOLD = 20)

LOGICAL*1 ITAVG

INTEGER I
INTEGER K
INTEGER NSOIL

```
DECLARE WORK ARRAYS NEEDED IN TRI-DIAGONAL IMPLICIT SOLVER
```

```
REAL AI(NSOLD)
REAL BI(NSOLD)
REAL CI(NSOLD)
```

```
DECLARATIONS
```

```
REAL BEXP
REAL CAIR
REAL CH2O
REAL CICE
REAL CSOIL
REAL DDZ
REAL DDZ2
REAL DENOM
REAL DF1
REAL DF1N
REAL DF1K
REAL DT
REAL DTSDZ
REAL DTSDZ2
REAL F1
REAL HCPCT
REAL PSISAT
REAL QUARTZ
REAL QTOT
REAL RHSTS(NSOIL)
REAL SSOIL
REAL SICE
REAL SMC(NSOIL)
REAL SH2O(NSOIL)
REAL SMCMAX
REAL SNKSR
REAL STC(NSOIL)
REAL TO
REAL TAVG
REAL TBK
REAL TBK1
REAL TBOT
REAL ZBOT
REAL TSNSR
REAL TSURF
REAL YY
REAL ZSOIL(NSOIL)
```

```
REAL ZZ1
```

```
PARAMETER(T0 = 273.15)
```

```
SET SPECIFIC HEAT CAPACITIES OF AIR, WATER, ICE, SOIL MINERAL
```

```
PARAMETER(CAIR = 1004.0)
```

```
PARAMETER(CH2O = 4.2E6)
```

```
PARAMETER(CICE = 2.106E6)
```

```
NOTE: CSOIL NOW SET IN ROUTINE REDPRM AND PASSED IN
```

```
PARAMETER(CSOIL = 1.26E6)
```

```
INITIALIZE LOGICAL FOR SOIL LAYER TEMPERATURE AVERAGING.
```

```
ITAVG = .TRUE.
```

```
ITAVG = .FALSE.
```

```
BEGIN SECTION FOR TOP SOIL LAYER
```

```
CALC THE HEAT CAPACITY OF THE TOP SOIL LAYER
```

```
HCPCT = SH2O(1)*CH2O + (1.0-SMCMAX)*CSOIL + (SMCMAX-SMC(1))*CAIR &
+ (SMC(1) - SH2O(1))*CICE
```

```
CALC THE MATRIX COEFFICIENTS AI, BI, AND CI FOR THE TOP LAYER
```

```
DDZ = 1.0 / ( -0.5 * ZSOIL(2) )
```

```
AI(1) = 0.0
```

```
CI(1) = (DF1 * DDZ) / (ZSOIL(1) * HCPCT)
```

```
BI(1) = -CI(1) + DF1 / (0.5 * ZSOIL(1) * ZSOIL(1)*HCPCT*ZZ1)
```

```
CALCULATE THE VERTICAL SOIL TEMP GRADIENT BTWN THE 1ST AND 2ND SOIL
LAYERS. THEN CALCULATE THE SUBSURFACE HEAT FLUX. USE THE TEMP
GRADIENT AND SUBSFC HEAT FLUX TO CALC "RIGHT-HAND SIDE TENDENCY
TERMS", OR "RHSTS", FOR TOP SOIL LAYER.
```

```
DTSDZ = (STC(1) - STC(2)) / (-0.5 * ZSOIL(2))
```

```
SSOIL = DF1 * (STC(1) - YY) / (0.5 * ZSOIL(1) * ZZ1)
```

```
RHSTS(1) = (DF1 * DTSDZ - SSOIL) / (ZSOIL(1) * HCPCT)
```

```
NEXT CAPTURE THE VERTICAL DIFFERENCE OF THE HEAT FLUX AT TOP AND
BOTTOM OF FIRST SOIL LAYER FOR USE IN HEAT FLUX CONSTRAINT APPLIED TO
```

POTENTIAL SOIL FREEZING/THAWING IN ROUTINE SNKSRC.

QTOT = SSOIL - DF1*DTSDZ

IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP):
SET TEMP "TSURF" AT TOP OF SOIL COLUMN (FOR USE IN FREEZING SOIL
PHYSICS LATER IN FUNCTION SUBROUTINE SNKSRC). IF SNOWPACK CONTENT IS
ZERO, THEN TSURF EXPRESSION BELOW GIVES TSURF = SKIN TEMP. IF
SNOWPACK IS NONZERO (HENCE ARGUMENT ZZ1=1), THEN TSURF EXPRESSION
BELOW YIELDS SOIL COLUMN TOP TEMPERATURE UNDER SNOWPACK. THEN
CALCULATE TEMPERATURE AT BOTTOM INTERFACE OF 1ST SOIL LAYER FOR USE
LATER IN FUNCTION SUBROUTINE SNKSRC

IF (ITAVG) THEN
TSURF = (YY + (ZZ1-1) * STC(1)) / ZZ1
CALL TBND (STC(1),STC(2),ZSOIL,ZBOT,1,NSOIL,TBK)
ENDIF

CALCULATE FROZEN WATER CONTENT IN 1ST SOIL LAYER.

SICE = SMC(1) - SH20(1)

IF FROZEN WATER PRESENT OR ANY OF LAYER-1 MID-POINT OR BOUNDING
INTERFACE TEMPERATURES BELOW FREEZING, THEN CALL SNKSRC TO
COMPUTE HEAT SOURCE/SINK (AND CHANGE IN FROZEN WATER CONTENT)
DUE TO POSSIBLE SOIL WATER PHASE CHANGE

IF ((SICE .GT. 0.) .OR. (TSURF .LT. TO) .OR. &
(STC(1) .LT. TO) .OR. (TBK .LT. TO)) THEN

IF (ITAVG) THEN
CALL TMPAVG(TAVG,TSURF,STC(1),TBK,ZSOIL,NSOIL,1)
ELSE
TAVG = STC(1)
ENDIF
TSNSR = SNKSRC (TAVG,SMC(1),SH20(1), &
ZSOIL,NSOIL,SMCMAX,PSISAT,BEXP,DT,1,QTOT)

RHSTS(1) = RHSTS(1) - TSNSR / (ZSOIL(1) * HCPCT)
ENDIF

THIS ENDS SECTION FOR TOP SOIL LAYER.

INITIALIZE DDZ2

```
-----  
      DDZ2 = 0.0
```

```
-----  
LOOP THRU THE REMAINING SOIL LAYERS, REPEATING THE ABOVE PROCESS  
(EXCEPT SUBSFC OR "GROUND" HEAT FLUX NOT REPEATED IN LOWER LAYERS)
```

```
-----  
      DF1K = DF1  
      DO K = 2,NSOIL
```

```
-----  
CALCULATE HEAT CAPACITY FOR THIS SOIL LAYER.
```

```
-----  
      HCPCT = SH20(K)*CH20 +(1.0-SMCMAX)*CSOIL +(SMCMAX-SMC(K))*CAIR &  
      + ( SMC(K) - SH20(K) )*CICE
```

```
-----  
      IF (K .NE. NSOIL) THEN
```

```
-----  
THIS SECTION FOR LAYER 2 OR GREATER, BUT NOT LAST LAYER.
```

```
-----  
CALCULATE THERMAL DIFFUSIVITY FOR THIS LAYER.
```

```
-----  
      CALL TDFCND (DF1N,SMC(K),QUARTZ,SMCMAX,SH20(K))
```

```
-----  
CALC THE VERTICAL SOIL TEMP GRADIENT THRU THIS LAYER
```

```
-----  
      DENOM = 0.5 * ( ZSOIL(K-1) - ZSOIL(K+1) )  
      DTSDZ2 = ( STC(K) - STC(K+1) ) / DENOM
```

```
-----  
CALC THE MATRIX COEF, CI, AFTER CALC'NG ITS PARTIAL PRODUCT
```

```
-----  
      DDZ2 = 2. / (ZSOIL(K-1) - ZSOIL(K+1))  
      CI(K) = -DF1N * DDZ2 / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)
```

```
-----  
IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP): CALCULATE  
TEMP AT BOTTOM OF LAYER.
```

```
-----  
      IF (ITAVG) THEN  
          CALL TBND (STC(K),STC(K+1),ZSOIL,ZBOT,K,NSOIL,TBK1)  
      ENDIF  
      ELSE
```

```
-----  
SPECIAL CASE OF BOTTOM SOIL LAYER: CALCULATE THERMAL DIFFUSIVITY FOR  
BOTTOM LAYER.
```

```
CALL TDFCND (DF1N,SMC(K),QUARTZ,SMCMAX,SH20(K))
```

```
CALC THE VERTICAL SOIL TEMP GRADIENT THRU BOTTOM LAYER.
```

```
DENOM = .5 * (ZSOIL(K-1) + ZSOIL(K)) - ZBOT
DTSDZ2 = (STC(K)-TBOT) / DENOM
```

```
SET MATRIX COEF, CI TO ZERO IF BOTTOM LAYER.
```

```
CI(K) = 0.
```

```
IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP): CALCULATE
TEMP AT BOTTOM OF LAST LAYER.
```

```
IF (ITAVG) THEN
    CALL TBND (STC(K),TBOT,ZSOIL,ZBOT,K,NSOIL,TBK1)
ENDIF
```

```
ENDIF
```

```
THIS ENDS SPECIAL LOOP FOR BOTTOM LAYER.
```

```
CALCULATE RHSTS FOR THIS LAYER AFTER CALC'NG A PARTIAL PRODUCT.
```

```
DENOM = ( ZSOIL(K) - ZSOIL(K-1) ) * HCPCT
RHSTS(K) = ( DF1N * DTSDZ2 - DF1K * DTSDZ ) / DENOM
QTOT = -1.0*DENOM*RHSTS(K)
SICE = SMC(K) - SH20(K)
```

```
IF ( (SICE .GT. 0.) .OR. (TBK .LT. TO) .OR. &
(STC(K) .LT. TO) .OR. (TBK1 .LT. TO) ) THEN
```

```
IF (ITAVG) THEN
    CALL TMPAVG(TAVG,TBK,STC(K),TBK1,ZSOIL,NSOIL,K)
ELSE
```

```
    TAVG = STC(K)
```

```
ENDIF
```

```
TSNSR = SNKSRC(TAVG,SMC(K),SH20(K),ZSOIL,NSOIL, &
SMCMAX,PSISAT,BEXP,DT,K,QTOT)
```

```
RHSTS(K) = RHSTS(K) - TSNSR / DENOM
```

```
ENDIF
```

```
CALC MATRIX COEFS, AI, AND BI FOR THIS LAYER.
```

```

-----  

      AI(K) = - DF1 * DDZ / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)  

      BI(K) = -(AI(K) + CI(K))  

-----  

      RESET VALUES OF DF1, DTSDZ, DDZ, AND TBK FOR LOOP TO NEXT SOIL LAYER.  

-----  

      TBK   = TBK1  

      DF1K  = DF1N  

      DTSDZ = DTSDZ2  

      DDZ   = DDZ2  

      END DO  

-----  

      END SUBROUTINE HRT  

-----  

      RETURN  

      END  

-----  

CCCC 4. PHYSICS SUBROUTINE ==> SUBROUTINE HRTICE CCCCCCCCCCCCCCCCCCCCC  

-----  

      SUBROUTINE HRTICE (RHSTS,STC,NSOIL,ZSOIL,YY,ZZ1,DF1,AI,BI,CI)  

-----  

      IMPLICIT NONE  

-----  

      SUBROUTINE HRTICE  

-----  

      CALCULATE THE RIGHT HAND SIDE OF THE TIME TENDENCY TERM OF THE SOIL  

      THERMAL DIFFUSION EQUATION IN THE CASE OF SEA-ICE PACK. ALSO TO  

      COMPUTE (PREPARE) THE MATRIX COEFFICIENTS FOR THE TRI-DIAGONAL MATRIX  

      OF THE IMPLICIT TIME SCHEME.  

-----  

      INTEGER NSOLD  

      PARAMETER(NSOLD = 20)  

-----  

      INTEGER K  

      INTEGER NSOIL  

-----  

      REAL AI(NSOLD)  

      REAL BI(NSOLD)  

      REAL CI(NSOLD)  

-----  

      REAL DDZ  

      REAL DDZ2  

      REAL DENOM  

      REAL DF1  

      REAL DTSDZ

```

```

REAL DTSDZ2
REAL HCPCT
REAL RHSTS(NSOIL)
REAL SSOIL
REAL STC(NSOIL)
REAL TBOT
REAL YY
REAL ZBOT
REAL ZSOIL(NSOIL)
REAL ZZ1

```

```
DATA TBOT /271.16/
```

```

SET A NOMINAL UNIVERSAL VALUE OF THE SEA-ICE SPECIFIC HEAT CAPACITY,
HCPCT = 1880.0*917.0.

```

```
PARAMETER(HCPCT = 1.72396E+6)
```

```

THE INPUT ARGUMENT DF1 IS A UNIVERSALLY CONSTANT VALUE OF SEA-ICE
THERMAL DIFFUSIVITY, SET IN ROUTINE SNOPAC AS DF1 = 2.2.

```

```

SET ICE PACK DEPTH. USE TBOT AS ICE PACK LOWER BOUNDARY TEMPERATURE
(THAT OF UNFROZEN SEA WATER AT BOTTOM OF SEA ICE PACK). ASSUME ICE
PACK IS OF N=NSOIL LAYERS SPANNING A UNIFORM CONSTANT ICE PACK
THICKNESS AS DEFINED BY ZSOIL(NSOIL) IN ROUTINE SFLX.

```

```
ZBOT = ZSOIL(NSOIL)
```

```
CALC THE MATRIX COEFFICIENTS AI, BI, AND CI FOR THE TOP LAYER
```

```

DDZ = 1.0 / ( -0.5 * ZSOIL(2) )
AI(1) = 0.0
CI(1) = (DF1 * DDZ) / (ZSOIL(1) * HCPCT)
BI(1) = -CI(1) + DF1/(0.5 * ZSOIL(1) * ZSOIL(1) * HCPCT * ZZ1)

```

```

CALC THE VERTICAL SOIL TEMP GRADIENT BTWN THE TOP AND 2ND SOIL LAYERS.
RECALC/ADJUST THE SOIL HEAT FLUX. USE THE GRADIENT AND FLUX TO CALC
RHSTS FOR THE TOP SOIL LAYER.

```

```

DTSDZ = ( STC(1) - STC(2) ) / ( -0.5 * ZSOIL(2) )
SSOIL = DF1 * ( STC(1) - YY ) / ( 0.5 * ZSOIL(1) * ZZ1 )
RHSTS(1) = ( DF1 * DTSDZ - SSOIL ) / ( ZSOIL(1) * HCPCT )

```

INITIALIZE DDZ2

DDZ2 = 0.0

LOOP THRU THE REMAINING SOIL LAYERS, REPEATING THE ABOVE PROCESS

DO K = 2,NSOIL

IF (K .NE. NSOIL) THEN

CALC THE VERTICAL SOIL TEMP GRADIENT THRU THIS LAYER.

DENOM = 0.5 * (ZSOIL(K-1) - ZSOIL(K+1))

DTSDZ2 = (STC(K) - STC(K+1)) / DENOM

CALC THE MATRIX COEF, CI, AFTER CALC'NG ITS PARTIAL PRODUCT.

DDZ2 = 2. / (ZSOIL(K-1) - ZSOIL(K+1))

CI(K) = -DF1 * DDZ2 / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)

ELSE

CALC THE VERTICAL SOIL TEMP GRADIENT THRU THE LOWEST LAYER.

DTSDZ2 = (STC(K)-TBOT)/(5 * (ZSOIL(K-1) + ZSOIL(K))-ZBOT)

SET MATRIX COEF, CI TO ZERO.

CI(K) = 0.

ENDIF

CALC RHSTS FOR THIS LAYER AFTER CALC'NG A PARTIAL PRODUCT.

DENOM = (ZSOIL(K) - ZSOIL(K-1)) * HCPCT

RHSTS(K) = (DF1 * DTSDZ2 - DF1 * DTSDZ) / DENOM

CALC MATRIX COEFS, AI, AND BI FOR THIS LAYER.

AI(K) = - DF1 * DDZ / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)

BI(K) = -(AI(K) + CI(K))

RESET VALUES OF DTSDZ AND DDZ FOR LOOP TO NEXT SOIL LYR.

```
-----  
      DTSDZ = DTSDZ2  
      DDZ    = DDZ2  
  
      END DO  
-----  
END SUBROUTINE HRTICE  
-----  
      RETURN  
      END  
  
CCCC 5. PHYSICS SUBROUTINE ==> SUBROUTINE HSTEP  CCCCCCCCCCCCCCCCCCCCC  
  
SUBROUTINE HSTEP (STCOUT,STCIN,RHSTS,DT,NSOIL,AI,BI,CI)  
  
IMPLICIT NONE  
  
-----  
SUBROUTINE HSTEP  
-----  
CALCULATE/UPDATE THE SOIL TEMPERATURE FIELD.  
-----  
      INTEGER NSOLD  
      PARAMETER(NSOLD = 20)  
  
      INTEGER K  
      INTEGER NSOIL  
  
      REAL AI(NSOLD)  
      REAL BI(NSOLD)  
      REAL CI(NSOLD)  
      REAL CIin(NSOLD)  
      REAL DT  
      REAL RHSTS(NSOIL)  
      REAL RHSTSin(NSOIL)  
      REAL STCIN(NSOIL)  
      REAL STCOUT(NSOIL)  
  
-----  
CREATE FINITE DIFFERENCE VALUES FOR USE IN ROSR12 ROUTINE  
-----  
      DO K = 1,NSOIL  
        RHSTS(K) = RHSTS(K) * DT  
        AI(K) = AI(K) * DT  
        BI(K) = 1. + BI(K) * DT  
        CI(K) = CI(K) * DT  
      END DO
```

COPY VALUES FOR INPUT VARIABLES BEFORE CALL TO ROSR12

```
DO K = 1,NSOIL
    RHSTSIn(K) = RHSTS(K)
END DO
DO K = 1,NSOLD
    CIin(K) = CI(K)
END DO
```

SOLVE THE TRI-DIAGONAL MATRIX EQUATION

```
CALL ROSR12(CI,AI,BI,CIin,RHSTSIn,RHSTS,NSOIL)
```

CALC/UPDATE THE SOIL TEMPS USING MATRIX SOLUTION

```
DO K = 1,NSOIL
    STCOUT(K) = STCIN(K) + CI(K)
END DO
```

END SUBROUTINE HSTEP

```
RETURN
END
```

CCCC 6. PHYSICS SUBROUTINE ==> SUBROUTINE NOPAC CCCCCCCCCCCCCCCCCCCCC

```
SUBROUTINE NOPAC(ETP,ETA,PRCP,SMC,SMCMAX,SMCWLT, &
                 SMCREF,SMCDRY,CMC,CMCMAX,NSOIL,DT,SHDFAC, &
                 SBETA,Q2,T1,SFCTMP,T24,TH2,FDOWN,F1,SSOIL, &
                 STC,EPSCA,BEXP,PC,RCH,RR,CFACTR, &
                 SH20,SLOPE,KDT,FRZFACT,PSISAT,ZSOIL, &
                 DKSAT,DWSAT,TBOT,ZBOT,RUNOFF1,RUNOFF2, &
                 RUNOFF3,EDIR,EC,ET,ETT,NROOT,ICE,RTDIS, &
                 QUARTZ,FXEXP,CSOIL, &
                 BETA,DRIP,DEW,FLX1,FLX2,FLX3)
```

IMPLICIT NONE

SUBROUTINE NOPAC

CALCULATE SOIL MOISTURE AND HEAT FLUX VALUES AND UPDATE SOIL MOISTURE
CONTENT AND SOIL HEAT CONTENT VALUES FOR THE CASE WHEN NO SNOW PACK IS
PRESENT.

```
INTEGER ICE
INTEGER NROOT
INTEGER NSOIL

REAL BEXP
REAL BETA
REAL CFACTR
REAL CMC
REAL CMCMAX
REAL CP
REAL CSOIL
REAL DEW
REAL DF1
REAL DKSAT
REAL DRIP
REAL DT
REAL DWSAT
REAL EC
REAL EDIR
REAL EPSCA
REAL ETA
REAL ETA1
REAL ETP
REAL ETP1
REAL ET(NSOIL)
REAL ETT
REAL FDOWN
REAL F1
REAL FXEXP
REAL FLX1
REAL FLX2
REAL FLX3
REAL FRZFACT
REAL KDT
REAL PC
REAL PRCP
REAL PRCP1
REAL PSISAT
REAL Q2
REAL QUARTZ
REAL RCH
REAL RR
REAL RTDIS(NSOIL)
REAL RUNOFF1
REAL RUNOFF2
REAL RUNOFF3
REAL SSOIL
```

```

REAL SBETA
REAL SFCTMP
REAL SHDFAC
REAL SH20(NSOIL)
REAL SIGMA
REAL SLOPE
REAL SMC(NSOIL)
REAL SMCDRY
REAL SMCMAX
REAL SMCREF
REAL SMCWLT
REAL STC(NSOIL)
REAL T1
REAL T24
REAL TBOT
REAL TH2
REAL YY
REAL YYNUM
REAL ZBOT
REAL ZSOIL(NSOIL)
REAL ZZ1

```

```

PARAMETER(CP = 1004.5)
PARAMETER(SIGMA = 5.67E-8)

```

EXECUTABLE CODE BEGINS HERE:

CONVERT ETP FROM KG M-2 S-1 TO MS-1 AND INITIALIZE DEW.

```

PRCP1 = PRCP * 0.001
ETP1 = ETP * 0.001
DEW = 0.0

```

```

IF (ETP .GT. 0.0) THEN

```

CONVERT PRCP FROM 'KG M-2 S-1' TO 'M S-1'.

```

CALL SMFLX (ETA1,SMC,NSOIL,CMC,ETP1,DT,PRCP1,ZSOIL, &
SH20,SLOPE,KDT,FRZFACT, &
SMCMAX,BEXP,PC,SMCWLT,DKSAT,DWSAT, &
SMCREF,SHDFAC,CMCMAX, &
SMCDRY,CFACTR,RUNOFF1,RUNOFF2,RUNOFF3, &
EDIR,EC,ET,ETT,SFCTMP,Q2,NROOT,RTDIS,FXEXP, &
DRIP)

```

CONVERT MODELED EVAPOTRANSPIRATION FM M S-1 TO KG M-2 S-1

```
-----  
      ETA = ETA1 * 1000.0  
  
      ELSE  
  
-----  
      IF ETP < 0, ASSUME DEW FORMS (TRANSFORM ETP1 INTO DEW AND REINITIALIZE  
      ETP1 TO ZERO).  
-----  
      DEW = -ETP1  
      ETP1 = 0.0  
  
-----  
      CONVERT PRCP FROM 'KG M-2 S-1' TO 'M S-1' AND ADD DEW AMOUNT.  
-----  
      PRCP1 = PRCP1 + DEW  
  
      CALL SMFLX (ETA1,SMC,NSOIL,CMC,ETP1,DT,PRCP1,ZSOIL, &  
                 SH20,SLOPE,KDT,FRZFACT, &  
                 SMCMAX,BEXP,PC,SMCWLT,DKSAT,DWSAT, &  
                 SMCREF,SHDFAC,CMCMAX, &  
                 SMCDRY,CFACTR,RUNOFF1,RUNOFF2,RUNOFF3, &  
                 EDIR,EC,ET,ETT,SFCTMP,Q2,NROOT,RTDIS,FXEXP, &  
                 DRIP)  
  
-----  
      CONVERT MODELED EVAPOTRANSPIRATION FROM 'M S-1' TO 'KG M-2 S-1'.  
-----  
      ETA = ETA1 * 1000.0  
  
      ENDIF  
  
-----  
      BASED ON ETP AND E VALUES, DETERMINE BETA  
-----  
      IF ( ETP .LE. 0.0 ) THEN  
          BETA = 0.0  
          IF ( ETP .LT. 0.0 ) THEN  
              BETA = 1.0  
              ETA = ETP  
          ENDIF  
      ELSE  
          BETA = ETA / ETP  
      ENDIF  
  
-----  
      GET SOIL THERMAL DIFFUXIVITY/CONDUCTIVITY FOR TOP SOIL LYR,  
      CALC. ADJUSTED TOP LYR SOIL TEMP AND ADJUSTED SOIL FLUX, THEN
```

CALL SHFLX TO COMPUTE/UPDATE SOIL HEAT FLUX AND SOIL TEMPS.

CALL TDFCND (DF1,SMC(1),QUARTZ,SMCMAX,SH20(1))

VEGETATION GREENNESS FRACTION REDUCTION IN SUBSURFACE HEAT FLUX
VIA REDUCTION FACTOR, WHICH IS CONVENIENT TO APPLY HERE TO THERMAL
DIFFUSIVITY THAT IS LATER USED IN HRT TO COMPUTE SUB SFC HEAT FLUX
(SEE ADDITIONAL COMMENTS ON VEG EFFECT SUB-SFC HEAT FLX IN
ROUTINE SFLX)

DF1 = DF1 * EXP(SBETA*SHDFAC)

COMPUTE INTERMEDIATE TERMS PASSED TO ROUTINE HRT (VIA ROUTINE
SHFLX BELOW) FOR USE IN COMPUTING SUBSURFACE HEAT FLUX IN HRT

YYNUM = FDOWN - SIGMA * T24
YY = SFCTMP + (YYNUM/RCH+TH2-SFCTMP-BETA*EPSCA) / RR
ZZ1 = DF1 / (-0.5 * ZSOIL(1) * RCH * RR) + 1.0

CALL SHFLX (SSOIL,STC,SMC,SMCMAX,NSOIL,T1,DT,YY,ZZ1,ZSOIL, &
TBOT,ZBOT,SMCWLT,PSISAT,SH20,BEXP,F1,DF1,ICE, &
QUARTZ,CSOIL)

SET FLX1 AND FLX3 (SNOPACK PHASE CHANGE HEAT FLUXES) TO ZERO SINCE
THEY ARE NOT USED HERE IN SNOPAC. FLX2 (FREEZING RAIN HEAT FLUX) WAS
SIMILARLY INITIALIZED IN THE PENMAN ROUTINE.

FLX1 = 0.0
FLX3 = 0.0

END SUBROUTINE NOPAC

RETURN
END

CCCC 7. PHYSICS SUBROUTINE ==> SUBROUTINE PENMAN CCCCCCCCCCCCCCCCCCCC

SUBROUTINE PENMAN (SFCTMP,SFCPRS,CH,T2V,TH2,PRCP,FDOWN,T24,SSOIL, &
Q2,Q2SAT,ETP,RCH,EPSCA,RR,SNOWNG,FRZGRA, &
DQSRT2,FLX2)

IMPLICIT NONE

SUBROUTINE PENMAN

CALCULATE POTENTIAL EVAPORATION FOR THE CURRENT POINT. VARIOUS PARTIAL SUMS/PRODUCTS ARE ALSO CALCULATED AND PASSED BACK TO THE CALLING ROUTINE FOR LATER USE.

LOGICAL*1 SNOWNG
LOGICAL*1 FRZGRA

REAL A
REAL BETA
REAL CH
REAL CP
REAL CPH20
REAL CPICE
REAL DELTA
REAL DQSDT2
REAL ELCP
REAL EPSCA
REAL ETP
REAL FDOWN
REAL FLX2
REAL FNET
REAL LSUBC
REAL LSUBF
REAL PRCP
REAL Q2
REAL Q2SAT
REAL R
REAL RAD
REAL RCH
REAL RHO
REAL RR
REAL SSOIL
REAL SFCPRS
REAL SFCTMP
REAL SIGMA
REAL T24
REAL T2V
REAL TH2

PARAMETER(CP = 1004.6)
PARAMETER(CPH20 = 4.218E+3)
PARAMETER(CPICE = 2.106E+3)
PARAMETER(R = 287.04)
PARAMETER(ELCP = 2.4888E+3)
PARAMETER(LSUBF = 3.335E+5)
PARAMETER(LSUBC = 2.501000E+6)

```
PARAMETER(SIGMA = 5.67E-8)
```

```
EXECUTABLE CODE BEGINS HERE:
```

```
FLX2 = 0.0
```

```
PREPARE PARTIAL QUANTITIES FOR PENMAN EQUATION.
```

```
DELTA = ELCP * DQSDT2
T24 = SFCTMP * SFCTMP * SFCTMP * SFCTMP
RR = T24 * 6.48E-8 / (SFCPRS * CH) + 1.0
RHO = SFCPRS / (R * T2V)
RCH = RHO * CP * CH
```

```
ADJUST THE PARTIAL SUMS / PRODUCTS WITH THE LATENT HEAT
EFFECTS CAUSED BY FALLING PRECIPITATION.
```

```
IF (.NOT. SNOWNG) THEN
  IF (PRCP .GT. 0.0) RR = RR + CPH20*PRCP/RCH
ELSE
  RR = RR + CPICE*PRCP/RCH
ENDIF
```

```
FNET = FDOWN - SIGMA*T24 - SSOIL
```

```
INCLUDE THE LATENT HEAT EFFECTS OF FRZNG RAIN CONVERTING TO ICE ON
IMPACT IN THE CALCULATION OF FLX2 AND FNET.
```

```
IF (FRZGRA) THEN
  FLX2 = -LSUBF * PRCP
  FNET = FNET - FLX2
ENDIF
```

```
FINISH PENMAN EQUATION CALCULATIONS.
```

```
RAD = FNET/RCH + TH2 - SFCTMP
A = ELCP * (Q2SAT - Q2)
EPSCA = (A*RR + RAD*DELTA) / (DELTA + RR)
ETP = EPSCA * RCH / LSUBC
```

```
END SUBROUTINE PENMAN
```

```
RETURN
END
```

CCCC 8. PHYSICS SUBROUTINE ==> SUBROUTINE ROSR12 CCCCCCCCCCCCCCCCCCCC

SUBROUTINE ROSR12 (P,A,B,C,D,DELTA,NSOIL)

IMPLICIT NONE

SUBROUTINE ROSR12

INVERT (SOLVE) THE TRI-DIAGONAL MATRIX PROBLEM SHOWN BELOW:

```
###                                     ### ### ### #####
#B(1), C(1), 0 , 0 , . . . , 0 # # # # #
#A(2), B(2), C(2), 0 , 0 , . . . , 0 # # # # #
# 0 , A(3), B(3), C(3), 0 , . . . , 0 # # # # # D(3) #
# 0 , 0 , A(4), B(4), C(4), . . . , 0 # # P(4) # # D(4) #
# 0 , 0 , 0 , A(5), B(5), . . . , 0 # # P(5) # # D(5) #
# .
# .
# .
# 0 , . . . , 0 , A(M-2), B(M-2), C(M-2), 0 # #P(M-2)# #D(M-2)#
# 0 , . . . , 0 , 0 , A(M-1), B(M-1), C(M-1) # #P(M-1)# #D(M-1)#
# 0 , . . . , 0 , 0 , 0 , A(M) , B(M) # # P(M) # # D(M) #
###                                     ### ### #####

```

INTEGER K
INTEGER KK
INTEGER NSOIL

REAL A(NSOIL)
REAL B(NSOIL)
REAL C(NSOIL)
REAL D(NSOIL)
REAL DELTA(NSOIL)
REAL P(NSOIL)

INITIALIZE EQN COEF C FOR THE LOWEST SOIL LAYER

C(NSOIL) = 0.0

SOLVE THE COEFS FOR THE 1ST SOIL LAYER

P(1) = -C(1) / B(1)
DELTA(1) = D(1) / B(1)

 SOLVE THE COEFS FOR SOIL LAYERS 2 THRU NSOIL

```

DO K = 2,NSOIL
  P(K) = -C(K) * ( 1.0 / (B(K) + A (K) * P(K-1)) )
  DELTA(K) = (D(K)-A(K)*DELTA(K-1))*(1.0/(B(K)+A(K)*P(K-1)))
END DO
  
```

 SET P TO DELTA FOR LOWEST SOIL LAYER

```
P(NSOIL) = DELTA(NSOIL)
```

 ADJUST P FOR SOIL LAYERS 2 THRU NSOIL

```

DO K = 2,NSOIL
  KK = NSOIL - K + 1
  P(KK) = P(KK) * P(KK+1) + DELTA(KK)
END DO
  
```

 END SUBROUTINE ROSR12

```

RETURN
END
  
```

CCCC 9. PHYSICS SUBROUTINE ==> SUBROUTINE SFCDIF CCCCCCCCCCCCCCCCCCCCC

SUBROUTINE SFCDIF (ZLM,ZO,THZO,THLM,SFCSPD,CZIL,AKMS,AKHS)

IMPLICIT NONE

 SUBROUTINE SFCDIF

CALCULATE SURFACE LAYER EXCHANGE COEFFICIENTS VIA ITERATIVE PROCESS.
SEE CHEN ET AL (1997, BLM)

```

REAL WWST, WWST2, G, VKRM, EXCM, BETA, BTG, ELFC, WOLD, WNEW
REAL PIHF, EPSU2, EPSUST, EPSIT, EPSA, ZTMIN, ZTMAX, HPBL, SQVISC
REAL RIC, RRIC, FHNEU, RFC, RFAC, ZZ, PSLMU, PSLMS, PSLHU, PSLHS
REAL XX, PSPMU, YY, PSPMS, PSPHU, PSPHS, ZLM, ZO, THZO, THLM
REAL SFCSPD, CZIL, AKMS, AKHS, ZILFC, ZU, ZT, RDZ, CXCH
REAL DTHV, DU2, BTGH, WSTAR2, USTAR, ZSLU, ZSLT, RLOGU, RLOGT
REAL RLMO, ZETALT, ZETALU, ZETAU, ZETAT, XLU4, XLT4, XU4, XT4
  
```

REAL XLU, XLT, XU, XT, PSMZ, SIMM, PSHZ, SIMH, USTARK, RLMN, RLMA
 CCREAL ZTFC

INTEGER ITRMX, ILECH, ITR

PARAMETER &
 (WWST=1.2,WWST2=WWST*WWST,G=9.8,VKRM=0.40,EXCM=0.001 &
 ,BETA=1./270.,BTG=BETA*G,ELFC=VKRM*BTG &
 ,WOLD=.15,WNEW=1.-WOLD,ITRMX=05,PIHF=3.14159265/2.)

PARAMETER &
 (EPSU2=1.E-4,EPSUST=0.07,EPSIT=1.E-4,EPSA=1.E-8 &
 ,ZTMIN=-5.,ZTMAX=1.,HPBL=1000.0 &
 ,SQVISC=258.2)

PARAMETER &
 (RIC=0.183,RRIC=1.0/RIC,FHNEU=0.8,RFC=0.191 &
 ,RFAC=RIC/(FHNEU*RFC*RFC))

NOTE: THE TWO CODE BLOCKS BELOW DEFINE FUNCTIONS

LECH'S SURFACE FUNCTIONS

PSLMU(ZZ)=-0.96*log(1.0-4.5*ZZ)
 PSLMS(ZZ)=ZZ*RRIC-2.076*(1.-1.)/(ZZ+1.)
 PSLHU(ZZ)=-0.96*log(1.0-4.5*ZZ)
 PSLHS(ZZ)=ZZ*RFAC-2.076*(1.-1.)/(ZZ+1.)

PAULSON'S SURFACE FUNCTIONS

PSPMU(XX)=-2.*log((XX+1.)*0.5)-log((XX*XX+1.)*0.5)+2.*ATAN(XX) &
 -PIHF
 PSPMS(YY)=5.*YY
 PSPHU(XX)=-2.*log((XX*XX+1.)*0.5)
 PSPHS(YY)=5.*YY

THIS ROUTINE SFCDIF CAN HANDLE BOTH OVER OPEN WATER (SEA, OCEAN) AND
 OVER SOLID SURFACE (LAND, SEA-ICE).

ILECH=0

ZTFC: RATIO OF ZOH/ZOM LESS OR EQUAL THAN 1

C.....ZTFC=0.1

CZIL: CONSTANT C IN Zilitinkevich, S. S.1995,:NOTE ABOUT ZT

```
ZILFC=-CZIL*VKRM*SQVISC
```

```
ZU=ZO
C.....ZT=ZO*ZTFC
RDZ=1./ZLM
CXCH=EXCM*RDZ
DTHV=THLM-THZ0
DU2=MAX(SFCSPD*SFCSPD,EPSU2)
```

```
BELJARS CORRECTION OF USTAR
```

```
BTGH=BTG*HPBL
cc If statements to avoid TANGENT LINEAR problems near zero
IF (BTGH*AKHS*DTHV .NE. 0.0) THEN
  WSTAR2=WWST2*ABS(BTGH*AKHS*DTHV)**(2./3.)
ELSE
  WSTAR2=0.0
ENDIF
USTAR=MAX(SQRT(AKMS*SQRT(DU2+WSTAR2)),EPSUST)
```

```
ZILITINKEVITCH APPROACH FOR ZT
```

```
ZT=EXP(ZILFC*SQRT(USTAR*ZO))*ZO
```

```
ZSLU=ZLM+ZU
ZSLT=ZLM+ZT
PRINT*, 'ZSLT=' ,ZSLT
PRINT*, 'ZLM=' ,ZLM
PRINT*, 'ZT=' ,ZT

RLOGU=log(ZSLU/ZU)
RLOGT=log(ZSLT/ZT)

RLMO=ELFC*AKHS*DTHV/USTAR**3
PRINT*, 'RLMO=' ,RLMO
PRINT*, 'ELFC=' ,ELFC
PRINT*, 'AKHS=' ,AKHS
PRINT*, 'DTHV=' ,DTHV
PRINT*, 'USTAR=' ,USTAR
```

```
DO ITR=1,ITRMX
```

```
1./MONIN-OBUKHOV LENGTH-SCALE
```

```

-----  

      ZETALT=MAX(ZSLT*RLMO,ZTMIN)  

      RLMO=ZETALT/ZSLT  

      ZETALU=ZSLU*RLMO  

      ZETAU=ZU*RLMO  

      ZETAT=ZT*RLMO  

  

      IF(ILECH.EQ.0) THEN  

        IF(RLMO.LT.0.)THEN  

          XLU4=1.-16.*ZETALU  

          XLT4=1.-16.*ZETALT  

          XU4 =1.-16.*ZETAU  

          XT4 =1.-16.*ZETAT  

  

          XLU=SQRT(SQRT(XLU4))  

          XLT=SQRT(SQRT(XLT4))  

          XU =SQRT(SQRT(XU4))  

          XT =SQRT(SQRT(XT4))  

  

          PSMZ=PSPMU(XU)  

          PRINT*, '-----1-----'  

          PRINT*, 'PSMZ=' ,PSMZ  

          PRINT*, 'PSPMU(ZETAU)=' ,PSPMU(ZETAU)  

          PRINT*, 'XU=' ,XU  

          PRINT*, '-----'  

            SIMM=PSPMU(XLU)-PSMZ+RLOGU  

            PSHZ=PSPHU(XT)  

            SIMH=PSPHU(XLT)-PSHZ+RLOGT  

        ELSE  

          ZETALU=MIN(ZETALU,ZTMAX)  

          ZETALT=MIN(ZETALT,ZTMAX)  

          PSMZ=PSPMS(ZETAU)  

          PRINT*, '-----2-----'  

          PRINT*, 'PSMZ=' ,PSMZ  

          PRINT*, 'PSPMS(ZETAU)=' ,PSPMS(ZETAU)  

          PRINT*, 'ZETAU=' ,ZETAU  

          PRINT*, '-----'  

            SIMM=PSPMS(ZETALU)-PSMZ+RLOGU  

            PSHZ=PSPHS(ZETAT)  

            SIMH=PSPHS(ZETALT)-PSHZ+RLOGT  

        ENDIF  

      ELSE  

-----  

LECH'S FUNCTIONS  

-----  

      IF(RLMO.LT.0.)THEN  

        PSMZ=PSLMU(ZETAU)  

      PRINT*, '-----3-----'

```

```

PRINT*, 'PSMZ=' ,PSMZ
PRINT*, 'PSLMU(ZETAU)=' ,PSLMU(ZETAU)
PRINT*, 'ZETAU=' ,ZETAU
PRINT*, '-----,
          SIMM=PSLMU(ZETALU)-PSMZ+RLOGU
          PSHZ=PSLHU(ZETAT)
          SIMH=PSLHU(ZETALT)-PSHZ+RLOGT
ELSE
          ZETALU=MIN(ZETALU,ZTMAX)
          ZETALT=MIN(ZETALT,ZTMAX)

          PSMZ=PSLMS(ZETAU)
PRINT*, '-----4-----,
PRINT*, 'PSMZ=' ,PSMZ
PRINT*, 'PSLMS(ZETAU)=' ,PSLMS(ZETAU)
PRINT*, 'ZETAU=' ,ZETAU
PRINT*, '-----,
          SIMM=PSLMS(ZETALU)-PSMZ+RLOGU
          PSHZ=PSLHS(ZETAT)
          SIMH=PSLHS(ZETALT)-PSHZ+RLOGT
ENDIF
ENDIF
-----
```

BELJAARS CORRECTION FOR USTAR

```
USTAR=MAX(SQRT(AKMS*SQRT(DU2+WSTAR2)),EPSUST)
```

ZILITINKEVITCH FIX FOR ZT

```
ZT=EXP(ZILFC*SQRT(USTAR*Z0))*Z0
```

```
ZSLT=ZLM+ZT
RLOGT=log(ZSLT/ZT)
```

```
USTARK=USTAR*VKRM
AKMS=MAX(USTARK/SIMM,CXCH)
AKHS=MAX(USTARK/SIMH,CXCH)
```

IF STATEMENTS TO AVOID TANGENT LINEAR PROBLEMS NEAR ZERO

```
IF (BTGH*AKHS*DTHV .NE. 0.0) THEN
  WSTAR2=WWST2*ABS(BTGH*AKHS*DTHV)**(2./3.)
ELSE
  WSTAR2=0.0
ENDIF
RLMN=ELFC*AKHS*DTHV/USTAR**3
```

```

RLMA=RLMO*WOLD+RLMN*WNEW
-----
IF(ABS((RLMN-RLMO)/RLMA).LT.EPSIT)      GO TO 110
-----
RLMO=RLMA
-----
END DO

PRINT*, '-----'
PRINT*, 'SFCDIF OUTPUT ! ! ! ! ! ! ! ! ! ! ! ! '
PRINT*, 'ZLM=' ,ZLM
PRINT*, 'ZO=' ,ZO
PRINT*, 'THZO=' ,THZO
PRINT*, 'THLM=' ,THLM
PRINT*, 'SFCSPD=' ,SFCSPD
PRINT*, 'CZIL=' ,CZIL
PRINT*, 'AKMS=' ,AKMS
PRINT*, 'AKHS=' ,AKHS
PRINT*, '-----'

-----
END SUBROUTINE SFCDIF
-----
RETURN
END

```

CCCC 10. PHYSICS SUBROUTINE ==> SUBROUTINE SFLX CCCCCCCCCCCCCCCCCCCCC

INTERFACE:

```

SUBROUTINE SFLX (ntl, &
ICE,DT,ZLVL,NSOIL,SLDPHT, &
LWDN,SOLDN,SFCPRS,PRCP,SFCTMP,Q2,SFCSPD, &
TH2,Q2SAT,DQSDT2, &
SLOPE,SHDFAC,SHDMIN,PTU,ALB,SNOALB, &
RSMIN,RGL,HS,SNUP,ZO,XLAI,NROOT, &
PSISAT,BEXP,DKSAT,SMCMAX,QUARTZ,DWSAT, &
SMCWLT,SMCREF,SMCDRY,F1,KDT,FRZX,FRZFACT,TBOT, &
CMC,T1,STC,SMC,SH20,SNOWH,SNEQV,ALBEDO,CH,CM, &
EVP,ETA,SHEAT, &
EC,EDIR,ET,ETT,ESNOW,DRIP,DEW, &
BETA,ETP,SSOIL, &
FLX1,FLX2,FLX3, &
SNOMLT,SNCVR, &
RUNOFF1,RUNOFF2,RUNOFF3, &
RC,PC,RCS,RCT,RCQ,RCSOIL, &
SOILW,SOILT,SOILM)

```

1.38.2 noahrst.F90 (Source File: noahrst.F90)

This program reads restart files for Noah. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA
              initial conditions if necessary. This is controlled with
              local variable NOAAIC. Normally set to 0 in this subroutine
              but set to 1 if want to use Dag's NOAA IC's. Changed output
              directory structure, and commented out if-then check so that
              directory is always made.
28 Apr 2002: Kristi Arsenault; Added NOAH LSM into LDAS
28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
              and put SMC, SH20, STC limit for GDAS and GEOS forcing.

```

RESTART FILE FORMAT(fortran sequential binary):

YR,MO,DA,HR,MN,SS,VCLASS,NCH	!Restart time,Veg class,no.tiles, no.soil lay
TILE(NCH)%COL	!Grid Col of Tile
TILE(NCH)%ROW	!Grid Row of Tile
TILE(NCH)%FGRD	!Fraction of Grid covered by tile
TILE(NCH)%VEGT	!Vegetation Type of Tile
NOAH(NCH)%STATES	!Model States in Tile Space

INTERFACE:

```
subroutine noahrst
```

USES:

```

use lisdrv_module, only : lis, grid, tile
use noah_varder, only : noahdrv
use time_module
USE noah_varder      ! NOAH tile variables
use time_manager
use tile_spmdMod

```

CONTENTS:

```

!-----
! Read Active Archive File
!-----
print*, 'DBG: noahrst -- in noahrst', ',iam,')
if(masterproc) then
  IF(LIS%0%STARTCODE.EQ.1)THEN
    allocate(tmptile(lis%d%nch))
    OPEN(40,FILE=noahdrv%NOAH_RFILE,FORM='unformatted')

```

```

call timemgr_read_restart(40)
call timemgr_restart()
call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
call updatetime(lis%t) !Updates LIS variables.
WRITE(*,*)'NOAH Restart File Used: ',noahdrv%NOAH_RFILE
READ(40) VCLASS,NC,NR,NCH !Time, veg class, no. tiles
!-----
! Check for Vegetation Class Conflict
!-----
IF(VCLASS.NE.LIS%P%VCLASS)THEN
    WRITE(*,*)noahdrv%NOAH_RFILE,' Vegetation class conflict'
    call endrun
ENDIF
!-----
! Check for Grid Space Conflict
!-----
IF(NC.NE.LIS%D%LNC.OR.NR.NE.LIS%D%LNR)THEN
    WRITE(*,*)noahdrv%NOAH_RFILE,'Grid space mismatch - NOAH HALTED'
    call endrun
ENDIF
!-----
! Transfer Restart tile space to LIS tile space
!-----
IF(NCH.NE.LIS%D%NCH)THEN
    WRITE(*,*)'Restart Tile Space Mismatch, Halting..'
    call endrun
endif
READ(40) noah%T1           !NOAH Skin Temperature (K)
READ(40) noah%CMC          !NOAH Canopy Water Content
READ(40) noah%SNOWH         !NOAH Actual Snow Depth (m)
READ(40) noah%SNEQV        !NOAH Water Equivalent Snow Depth (m)
DO L=1,4
    READ(40) TMPTILE !NOAH Soil Layer Temp (4 layers)
    noah%STC(L)=TMPTILE
ENDDO
DO L=1,4
    READ(40) TMPTILE !NOAH Total soil moist. (4 layers)
    noah%SMC(L)=TMPTILE
ENDDO
DO L=1,4
    READ(40) TMPTILE !NOAH Liquid-only soil moist. (4 layers)
    noah%SH2O(L)=TMPTILE
ENDDO
READ(40) noah%CH           !NOAH Sfc Exchange Coef. for Heat/Moisture
READ(40) noah%CM           !NOAH Sfc Exchange Coef. for Momentum
close(40)
deallocate(tmptile)

```

```
        endif
    endif
```

1.38.3 noah_scatter.F90 (Source File: noah_scatter.F90)

Distributes noah tiles on to compute nodes

REVISION HISTORY:

Apr 2003 ; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine noah_scatter()
```

USES:

```
use tile_spmdMod
use noah_varder
use noahpardef_module
```

CONTENTS:

```
call MPI_SCATTERV(noah,di_array,displs, &
                  MPI_NOAH_STRUCT,noah,di_array(iam),MPI_NOAH_STRUCT, &
                  0,MPI_COMM_WORLD,ierr)
```

1.38.4 noah_setup.F90 (Source File: noah_setup.F90)

Complete the setup routines for noah

REVISION HISTORY:

4 Nov. 1999: Paul Houser; Initial Code

28 Apr. 2002: K. Arsenault; Modified to NOAH LSM 2.5 code to LDAS

INTERFACE:

```
subroutine noah_setup()
```

USES:

```
use lisdrv_module, only: lis,tile
use noah_varder
use spmdMod, only : masterproc, npes
```

CONTENTS:

```

#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endiff
  call setnoahp()
  call noah_gfrac()
  call noah_alb()
  call noah_coldstart()
  do t=1,lis%d%nch
    noah(t)%swnet = 0
    noah(t)%lwnet = 0
    noah(t)%qle = 0
    noah(t)%qh = 0
    noah(t)%qg = 0
    noah(t)%snowf = 0
    noah(t)%rainf = 0
    noah(t)%evap = 0
    noah(t)%qs = 0
    noah(t)%qsb = 0
    noah(t)%qsm = 0
    noah(t)%swe = 0
    noah(t)%soilmoist1 = 0
    noah(t)%soilmoist2 = 0
    noah(t)%soilmoist3 = 0
    noah(t)%soilmoist4 = 0
    noah(t)%soilwet = 0
    noah(t)%tveg = 0
    noah(t)%esoil = 0
    noah(t)%rootmoist =0
    noah(t)%soilm_prev = 0
    noah(t)%swe_prev = 0
    noah(t)%count = 0
  enddo
#endif ( ! defined OPENDAP )
  endif
  if ( npes > 1 ) then
    call noah_scatter
  endif
#endiff

```

1.38.5 noah_singlegather.F90 (Source File: noah_singlegather.F90)

Gather single variable for output

REVISION HISTORY:

Apr 2003 ; Sujay Kumar, Initial Code

INTERFACE:

```
\bigskip{\em USES:}

\begin{verbatim}    use lisdrv_module, only : lis
    use tile_spmdMod
    use noah_varder
    use noahpardef_module
    IMPLICIT NONE
```

ARGUMENTS:

```

integer :: index           ! Index of Noah variable
real    :: var(lis%d%glbnch) ! Noah variable being gathered

```

CONTENTS:

```

        noah(t)%smc(3)*1000.0*0.6 + &
        noah(t)%smc(4)*1000.0 - noah(t)%soilm_prev
case(13)
    var_temp(t) = noah(t)%sneqv*1000.0-noah(t)%swe_prev
case(14)
    var_temp(t) = noah(t)%avgsurft
case(15)
    var_temp(t) = noah(t)%albedo
case(16)
    var_temp(t) = noah(t)%swe/float(noah(t)%count)
case(17)
    var_temp(t) = noah(t)%soilmoist1/float(noah(t)%count)
case(18)
    var_temp(t) = noah(t)%soilmoist2/float(noah(t)%count)
case(19)
    var_temp(t) = noah(t)%soilmoist3/float(noah(t)%count)
case(20)
    var_temp(t) = noah(t)%soilmoist4/float(noah(t)%count)
case(21)
    var_temp(t) = noah(t)%soilwet/float(noah(t)%count)
case(22)
    var_temp(t) = noah(t)%tveg/float(noah(t)%count)
case(23)
    var_temp(t) = noah(t)%esoil/float(noah(t)%count)
case(24)
    var_temp(t) = noah(t)%rootmoist/float(noah(t)%count)
case(25)
    if(lis%o%wfor.eq.1) then
        var_temp(t) = sqrt(noah(t)%forcing(5)*noah(t)%forcing(5)+ &
                      noah(t)%forcing(6)*noah(t)%forcing(6))
    endif
case(26)
    if(lis%o%wfor.eq.1) then
        if(noah(t)%forcing(1) < 273.15) then
            var_temp(t) = 0.0
        else
            var_temp(t) = noah(t)%forcing(8)
        endif
    endif
case(27)
    if(lis%o%wfor.eq.1) then
        if(noah(t)%forcing(1) < 273.15) then
            var_temp(t) = noah(t)%forcing(8)
        else
            var_temp(t) = 0.0
        endif
    endif
case(28)

```

```

        if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(1)
        endif
    case(29)
        if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(2)
        endif
    case(30)
        if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(7)
        endif
    case(31)
        if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(3)
        endif
    case(32)
        if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(4)
        endif
    end select
enddo

call MPI_GATHERV(var_temp(1:di_array(iam)), &
                 di_array(iam), &
                 MPI_REAL,var,di_array,displs,MPI_REAL, &
                 0,MPI_COMM_WORLD, ierr)

```

1.38.6 noah_singleout.F90 (Source File: noah_singleout.F90)

Write output file for a single noah variable

REVISION HISTORY:

14 Jun 2002 Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine noah_singleout (ld,tile,gindex, var_array, index)
```

USES:

```

use lis_module      ! LDAS non-model-specific 1-D variables
use tile_module     ! LDAS non-model-specific tile variables
use time_manager, only : get_nstep
use noah_varder, only : noahdrv

implicit none

```

ARGUMENTS:

```

type (lisdec) :: ld      !data structure for lis domain specific variables
type (tileddec) :: tile(ld%d%glnch) !tile array for the modeled domain
integer         :: gindex(ld%d%lnc, ld%d%lnr) !2-d array for mapping from 2d to 1d
real            :: var_array(ld%d%glnch) !array of variable that is being output
integer         :: index   !Index of the output variable in the ALMA list.

```

CONTENTS:

```

!-----
! Test to see if output writing interval has been reached
!-----
IF(MOD(LD%T%GMT, noahdrv%WRITEINTN).EQ.0)THEN
  noahdrv%NUMOUTNH=noahdrv%NUMOUTNH+1
!-----
! Generate directory structure and file names for NOAH output
!-----
length = len(trim(vname1(index)))
WRITE(UNIT=temp1, FMT='(A10)') VNAME1(index)
READ(UNIT=temp1,FMT='(10A1)') (FVARNAME(I), I=1,length)
WRITE(UNIT=temp1,FMT='(I4,I2,I2)')LD%T%YR,LD%T%MO,LD%T%DA
READ(UNIT=temp1,FMT='(8A1)') FTIME
DO I=1,8
  IF(FTIME(I).EQ.(' '))FTIME(I)='0'
ENDDO
WRITE(UNIT=temp1,FMT='(I4)')LD%T%YR
READ(UNIT=temp1,FMT='(8A1)')FTIMEC
DO I=1,4
  IF(FTIMEC(I).EQ.(' '))FTIMEC(I)='0'
ENDDO

WRITE(UNIT=temp1,FMT='(A7,I3,A1)') '/LDAS.E',LD%O%EXPCODE,'.
READ(UNIT=temp1,FMT='(80A1)') (FNAME(I),I=1,11)
DO I=1,11
  IF(FNAME(I).EQ.(' '))FNAME(I)='0'
ENDDO

WRITE(UNIT=temp1,FMT='(A40)') LD%O%ODIR
READ(UNIT=temp1,FMT='(40A1)') (FBASE(I),I=1,40)
C=0
DO I=1,40
  IF(FBASE(I).EQ.(' ').AND.C.EQ.0)C=I-1
ENDDO

WRITE(UNIT=temp1,FMT='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
LD%O%EXPCODE,'/NOAH/', &
LD%T%YR,'/ ',LD%T%YR,LD%T%MO,LD%T%DA
READ(UNIT=temp1,FMT='(80A1)') (FYRMODIR(I),I=1,26)

```

```

DO I=1,26
  IF(FYRMODIR(I).EQ.(' '))FYRMODIR(I)='0'
ENDDO

WRITE(UNIT=temp1,FMT='(A9)')'mkdir -p '
READ(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9)

WRITE(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
  (FYRMODIR(I),I=1,26)
READ(UNIT=temp1,FMT='(A80)')MKFYRMO
!-----
!  Make the directories for the NOAH output data files
!-----
CALL SYSTEM(MKFYRMO)
!-----
! Generate file name for BINARY output
!-----
IF(LD%0%WOUT.EQ.1) THEN
  WRITE(UNIT=FBINNAME, FMT='(I4,I2,I2,I2)') LD%T%YR,LD%T%MO, &
    LD%T%DA,LD%T%HR
  READ(UNIT=FBINNAME,FMT='(10A1)') FTIMEB
  DO I=1,10
    IF(FTIMEB(I).EQ.(' '))FTIMEB(I)='0'
  ENDDO
  WRITE(UNIT=FBINNAME,FMT='(A9)') '.NOAHgbin'
  READ(UNIT=FBINNAME,FMT='(80A1)')(FSUBGB(I),I=1,9)
  WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
    (FYRMODIR(I),I=1,26), &
    (FNAME(I),I=1,11),(FTIMEB(I),I=1,10), &
    (FVARNAME(I), I=1,length),(FSUBGB(I),I=1,9)
  READ(UNIT=FBINNAME,FMT='(A80)')FILENGB
!-----
! Open statistical output file
!-----
IF(noahdrv%NOAHopen.EQ.0)THEN
  FILE='NOAHstats.dat'
  CALL OPENFILE(NAME,LD%0%ODIR,LD%0%EXPCODE,FILE)
  IF(LD%0%STARTCODE.EQ.1)THEN
    OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='UNKNOWN', &
      POSITION='APPEND')
  ELSE
    OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='REPLACE')
  ENDIF
  noahdrv%NOAHopen=1
ENDIF

WRITE(65,996)' Statistical Summary of NOAH Output for: ', &
  LD%T%MO,'/',LD%T%DA,'/',LD%T%YR,LD%T%HR,:',LD%T%MN,:',LD%T%SS

```

```

996      FORMAT(A47,I2,A1,I2,A1,I4,1X,I2,A1,I2,A1,I2)
         WRITE(65,*)
         WRITE(65,997)
997      FORMAT(T27,'Mean',T41,'StDev',T56,'Min',T70,'Max')
      ENDIF
!-----
! Write output in HDF and binary (if WBIN=1) format
!-----
      IF(LD%WOUT.EQ.1) then
         OPEN(58,file=FILENGB,FORM='UNFORMATTED')
      endif
      if(ld%wout.eq.1) then
         call t2gr(var_array,gtmp,ld%d%glbngrid, &
                    ld%d%glbnch, tile);
         write(58) gtmp
         call stats(var_array,ld%d%undef,ld%d%glbnch, vmean, &
                    vstdev, vmin, vmax);
         write(65,999) vname(index),vmean, vstdev, vmin, vmax
      endif

998 FORMAT(1X,A18,4E14.3)
999 FORMAT(1X,A18,4F14.3)
      endif

```

1.38.7 noah_soiltype.F90 (Source File: noah_soil_typ.F90)

This subroutine uses the percentages of sand and clay derived from the global soils dataset of Reynolds, Jackson, and Rawls [1999], to convert to Zobler soil class values to be used in NOAH LSM v2.5 in LDAS. (Original code by Matt Rodell, 3/7/01)

28 Apr 2002, K Arsenault: Added NOAH LSM to LDAS

INTERFACE:

```
subroutine soiltype(nc,nr,sand,clay,soiltyp)
```

CONTENTS:

```

do j=1,nr
  do i=1,nc
    if (clay(i,j) .lt. 0.00) then
      soiltyp(i,j) = -99
    else
      cl = clay(i,j)
      sa = sand(i,j)
    endif
!-----
!     identify texture class.

```

```
!-----
      if (cl .lt. 0.23) then
          if (sa .lt. 0.50) then
              soiltyp(i,j) = 8           ! loam
          else
              if (sa .lt. 0.75) then
                  soiltyp(i,j) = 4       ! sandy loam
              else
                  soiltyp(i,j) = 1       ! loamy sand
              end if
          end if
      else
          if (cl .lt. 0.28) then
              if (sa .lt. 0.45) then
                  soiltyp(i,j) = 8           ! loam
              else
                  soiltyp(i,j) = 7       ! sandy clay loam
              endif
          else
              if (cl .lt. 0.37) then
                  if (sa .lt. 0.2) then
                      soiltyp(i,j) = 2       ! silty clay loam
                  else
                      if (sa .lt. 0.43) then
                          soiltyp(i,j) = 6       ! clay loam
                      else
                          soiltyp(i,j) = 7       ! sandy clay loam
                      end if
                  end if
              else
                  if (cl .lt. 0.41) then
                      if (sa .lt. 0.2) then
                          soiltyp(i,j) = 2       ! silty clay loam
                      else
                          if (sa .lt. 0.43) then
                              soiltyp(i,j) = 6       ! clay loam
                          else
                              soiltyp(i,j) = 5       ! sandy clay
                          end if
                      end if
                  else
                      if (sa .lt. 0.43) then
                          soiltyp(i,j) = 3       ! light clay
                      else
                          soiltyp(i,j) = 5       ! sandy clay
                      end if
                  end if
              end if
          end if
      end if
```

```

    end if
end if
end do
end do

```

1.38.8 noah_tileout.F90 (Source File: noah_tileout.F90)

LIS NOAH data writer: Writes output in tile space

REVISION HISTORY:

02 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
subroutine noah_tileout(ld,tile,ftn,ftn_stats)
```

USES:

```

use lis_module
use tile_module
use noah_varder

implicit none

type(lisdec) :: ld
type(tiledesc) :: tile
integer :: ftn,ftn_stats

```

CONTENTS:

```

do t=1,ld%d%glbnch
  if(noah(t)%forcing(1) < 273.15) then
    rainf(t) = 0.0
    snowf(t) = noah(t)%forcing(8)
  else
    rainf(t) = noah(t)%forcing(8)
    snowf(t) = 0.0
  endif
enddo
!-----
! General Energy Balance Components
!-----
noah%swnet = noah%swnet/float(noah%count)
call t2gr(noah%swnet,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gttmp !Net shortwave radiation (surface) (W/m2)
call stats(noah%swnet,ld%d%udef,ld%d%glbnch,vmean, &

```

```

      vstdev,vmin, vmax)
write(ftn_stats,999) 'SWnet(W/m2)', &
      vmean,vstdev,vmin,vmax

noah%lwnet = noah%lwnet/float(noah%count)
call t2gr(noah%lwnet,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Net longwave radiation (surface) (W/m2)

call stats(noah%lwnet,ld%d%udef,ld%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,999) 'LWnet(W/m2)',&
      vmean,vstdev,vmin,vmax

noah%qle = noah%qle/float(noah%count)
call t2gr(noah%qle,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Latent Heat Flux (W/m2)

call stats(noah%qle,ld%d%udef,ld%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,999) 'Qle(W/m2)',&
      vmean,vstdev,vmin,vmax

noah%qh = noah%qh/float(noah%count)
call t2gr(noah%qh,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Sensible Heat Flux (W/m2)

call stats(noah%qh,ld%d%udef,ld%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,999) 'Qh(W/m2)',&
      vmean,vstdev,vmin,vmax

noah%qg = noah%qg/float(noah%count)
call t2gr(noah%qg,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Ground Heat Flux (W/m2)

call stats(noah%qg,ld%d%udef,ld%d%glbnch,vmean, &
      vstdev,vmin, vmax)
write(ftn_stats,999) 'Qg(W/m2)',&
      vmean,vstdev,vmin,vmax
!-----
! General Water Balance Components
!-----
noah%snowf = noah%snowf/float(noah%count)
call t2gr(noah%snowf,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Snowfall rate (kg/m2s)

call stats(noah%snowf,ld%d%udef,ld%d%glbnch,vmean, &
      vstdev,vmin, vmax)

```

```

write(ftn_stats,998) 'Snowf(kg/m2s)',&
vmean,vstdev,vmin,vmax

noah%rainf = noah%rainf/float(noah%count)
call t2gr(noah%rainf,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Snowfall rate (kg/m2s)

call stats(noah%rainf,ld%d%udef,ld%d%glbnch,vmean, &
vstdev,vmin, vmax)
write(ftn_stats,998) 'Rainf(kg/m2s)',&
vmean,vstdev,vmin,vmax

noah%evap = noah%evap/float(noah%count)
call t2gr(noah%evap,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Evapotranspiration (kg/m2s)???

call stats(noah%evap,ld%d%udef,ld%d%glbnch,vmean, &
vstdev,vmin, vmax)
write(ftn_stats,998) 'Evap(kg/m2s)',&
vmean,vstdev,vmin,vmax

noah%qs = noah%qs/float(noah%count)
call t2gr(noah%qs,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Surface Runoff(kg/m2s)

call stats(noah%qs,ld%d%udef,ld%d%glbnch,vmean, &
vstdev,vmin, vmax)
write(ftn_stats,998) 'Qs(kg/m2s)',&
vmean,vstdev,vmin,vmax

noah%qsb = noah%qsb/float(noah%count)
call t2gr(noah%qsb,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Subsurface Runoff (kg/m2s)???

call stats(noah%qsb,ld%d%udef,ld%d%glbnch,vmean, &
vstdev,vmin, vmax)
write(ftn_stats,998) 'Qsb(kg/m2s)',&
vmean,vstdev,vmin,vmax

noah%qsm = noah%qsm/float(noah%count)
call t2gr(noah%qsm,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp !Snowmelt (kg/m2s)

call t2gr(noah%smc(1)*1000.0*0.1+ &
noah%smc(2)*1000.0*0.3 + &
noah%smc(3)*1000.0*0.6 + &
noah%smc(4)*1000.0 -noah%soilm_prev, &
gtmp,ld%d%glbngrid, &

```

```

    ld%d%glbnch, tile)
write(ftn) gtmp          !DelSoilMoist
call stats(noah%smc(1)*1000.0*0.1+ &
           noah%smc(2)*1000.0*0.3 + &
           noah%smc(3)*1000.0*0.6 + &
           noah%smc(4)*1000.0 -noah%soilm_prev, &
           ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,999) 'DelSoilMoist(kg/m2s)', &
                     vmean,vstdev,vmin,vmax
call t2gr( noah%sneqv*1000.0-noah%swe_prev,gtmp,ld%d%glbngrid, &
           ld%d%glbnch, tile)
write(ftn) gtmp          !DelSWE
call stats( noah%sneqv*1000.0-noah%swe_prev, &
           ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,999) 'DelSWE(kg/m2s)', &
                     vmean,vstdev,vmin,vmax
!-----
! Surface State Variables
! missing snowT, vegT, baresoilT,RadT
!-----
call t2gr(noah%avgsurft,gtmp,ld%d%glbngrid,ld%d%glbnch, tile)
write(ftn) gtmp !Average Surface Temperature (K)

call stats(noah%avgsurft,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'AvgSurfT(K)',&
                     vmean,vstdev,vmin,vmax

call t2gr(noah%albedo,gtmp,ld%d%glbngrid,ld%d%glbnch, tile)
write(ftn) gtmp !Surface Albedo (-)

call stats(noah%albedo,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'Albedo(-)',&
                     vmean,vstdev,vmin,vmax

noah%swe= noah%swe/float(noah%count)
call t2gr(noah%swe,gtmp,ld%d%glbngrid,ld%d%glbnch, tile)
write(ftn) gtmp !Snow water equivalent (kg/m2)

call stats(noah%swe,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'SWE(kg/m2)',&
                     vmean,vstdev,vmin,vmax
!-----
!== Subsurface State Variables
!-----
noah%soilmoist1= noah%soilmoist1/float(noah%count)

```

```

call t2gr(noah%soilmoist1,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Soil water content for layer1 (kg/m2)

call stats(noah%soilmoist1,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist1(kg/m2)',&
                     vmean,vstdev,vmin,vmax

noah%soilmoist2= noah%soilmoist2/float(noah%count)
call t2gr(noah%soilmoist1,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Soil water content for layer2 (kg/m2)

call stats(noah%soilmoist2,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist2(kg/m2)',&
                     vmean,vstdev,vmin,vmax

noah%soilmoist3= noah%soilmoist3/float(noah%count)
call t2gr(noah%soilmoist1,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Soil water content for layer3 (kg/m2)

call stats(noah%soilmoist3,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist3(kg/m2)',&
                     vmean,vstdev,vmin,vmax

noah%soilmoist4= noah%soilmoist4/float(noah%count)
call t2gr(noah%soilmoist1,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Soil water content for layer4 (kg/m2)

call stats(noah%soilmoist4,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,999) 'SoilMoist4(kg/m2)',&
                     vmean,vstdev,vmin,vmax

noah%soilwet= noah%soilwet/float(noah%count)
call t2gr(noah%soilwet,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Total Soil Wetness (-)

call stats(noah%soilwet,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'SoilWet(-)',&
                     vmean,vstdev,vmin,vmax
!-----
!==== Evaporation Components
!-----

noah%tveg= noah%tveg/float(noah%count)
call t2gr(noah%tveg,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)

```

```

write(ftn) gtmp ! Vegetation transpiration (kg/m2s)

call stats(noah%tveg,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'TVeg(kg/m2s)',&
           vmean,vstdev,vmin,vmax

noah%esoil= noah%esoil/float(noah%count)
call t2gr(noah%esoil,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Bare soil evaporation (kg/m2s)

call stats(noah%esoil,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'ESoil(kg/m2s)',&
           vmean,vstdev,vmin,vmax

noah%rootmoist = noah%rootmoist/float(noah%count)
call t2gr(noah%rootmoist,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
write(ftn) gtmp ! Root zone soil moisture (kg/m2)

call stats(noah%rootmoist,ld%d%udef,ld%d%glbnch,vmean, &
           vstdev,vmin, vmax)
write(ftn_stats,998) 'RootMoist(kg/m2)',&
           vmean,vstdev,vmin,vmax

if(ld%o%wfor.eq.1) then
  call t2gr(sqrt(noah%forcing(5)*noah%forcing(5)+ &
                 noah%forcing(6)*noah%forcing(6)),gtmp,ld%d%glbngrid, &
                 ld%d%glbnch, tile)
  write(ftn) gtmp      !Wind
  call stats(sqrt(noah%forcing(5)*noah%forcing(5)+ &
                 noah%forcing(6)*noah%forcing(6)), &
                 ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
  write(ftn_stats,999) 'Wind(m/s)', &
           vmean,vstdev,vmin,vmax

  call t2gr(rainf, &
            gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp      !Rainf
  call stats(rainf, &
            ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
  write(ftn_stats,998) 'Rainf(kg/m2s)', &
           vmean,vstdev,vmin,vmax

  call t2gr(snowf, &
            gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
  write(ftn) gtmp      !Snowf
  call stats(snowf, &

```

```

1d%d%udef,1d%d%glbnch,vmean,vstdev,vmin, vmax)
write(ftn_stats,998) 'Snowf(kg/m2s)', &
vmean,vstdev,vmin,vmax

call t2gr(noah%forcing(1),gtmp,1d%d%glbngrid,1d%d%glbnch,tile)
write(ftn) gtmp          !Tair
call stats(noah%forcing(1),1d%d%udef,1d%d%glbnch,vmean,vstdev, &
vmin, vmax)
write(ftn_stats,999) 'Tair(K)', &
vmean,vstdev,vmin,vmax
call t2gr(noah%forcing(2),gtmp,1d%d%glbngrid,1d%d%glbnch,tile)
write(ftn) gtmp          !Qair
call stats(noah%forcing(2),1d%d%udef,1d%d%glbnch,vmean,vstdev, &
vmin, vmax)
write(ftn_stats,999) 'Qair(kg/kg)', &
vmean,vstdev,vmin,vmax
call t2gr(noah%forcing(7),gtmp,1d%d%glbngrid,1d%d%glbnch,tile)
write(ftn) gtmp          !PSurf
call stats(noah%forcing(7),1d%d%udef,1d%d%glbnch,vmean,vstdev, &
vmin, vmax)
write(ftn_stats,999) 'PSurf(Pa)', &
vmean,vstdev,vmin,vmax
call t2gr(noah%forcing(3),gtmp,1d%d%glbngrid,1d%d%glbnch,tile)
write(ftn) gtmp          !SWdown
call stats(noah%forcing(3),1d%d%udef,1d%d%glbnch,vmean,vstdev, &
vmin, vmax)
write(ftn_stats,999) 'SWdown (W/m2)', &
vmean,vstdev,vmin,vmax

call t2gr(noah%forcing(4),gtmp,1d%d%glbngrid,1d%d%glbnch,tile)
write(ftn) gtmp          !LWdown
call stats(noah%forcing(4),1d%d%udef,1d%d%glbnch,vmean,vstdev, &
vmin, vmax)
write(ftn_stats,999) 'LWdown(W/m2)', &
vmean,vstdev,vmin,vmax

endif
998 FORMAT(1X,A18,4E14.3)
999 FORMAT(1X,A18,4F14.3)

```

1.38.9 noah_totinit.F90 (Source File: noah_totinit.F90)

Initialize NOAH output arrays

REVISION HISTORY:

14 Jun 2002 Sujay Kumar Initial Specification

INTERFACE:

```
subroutine noah_totinit()
```

USES:

```
use noah_varder      ! NOAH LSM module
use tile_spmdMod
use lisdrv_module, only : lis
```

CONTENTS:

```
do t = 1, di_array(iam)
  if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
    noah(t)%soilm_prev=noah(t)%smc(1)*1000.0*0.1+ &
      noah(t)%smc(2)*1000.0*0.3 + &
      noah(t)%smc(3)*1000.0*0.6 + &
      noah(t)%smc(4)*1000.0
    noah(t)%swe_prev = noah(t)%sneqv*1000.0
  endif
enddo
do t = 1, di_array(iam)
  noah(t)%swnet = 0
  noah(t)%lwnet = 0
  noah(t)%qle = 0
  noah(t)%qh = 0
  noah(t)%qg = 0
  noah(t)%snowf = 0
  noah(t)%rainf = 0
  noah(t)%evap = 0
  noah(t)%qs = 0
  noah(t)%qsb = 0
  noah(t)%qsm = 0
  noah(t)%swe = 0
  noah(t)%soilmoist1 = 0
  noah(t)%soilmoist2 = 0
  noah(t)%soilmoist3 = 0
  noah(t)%soilmoist4 = 0
  noah(t)%soilwet = 0
  noah(t)%tveg = 0
  noah(t)%esoil = 0
  noah(t)%rootmoist =0
  noah(t)%count = 0
enddo
```

1.38.10 noah_varder.F90 (Source File: noah_varder.F90)

Module for 1-D NOAH land model driver variable initialization

REVISION HISTORY:

Apr 2003; Sujay Kumar, Initial Code

INTERFACE:

```
module noah_varder
```

USES:

```
use noah_module
use tile_spmdMod
use noahpardef_module
use noahdrv_module
```

1.38.11 noah_varder_ini (Source File: noah_varder.F90)

Reads in runtime noah parameters, allocates memory for variables

INTERFACE:

```
subroutine noah_varder_ini(nch)
```

USES:

```
#if ( defined OPENDAP )
    use opendap_module
#endif
```

CONTENTS:

```
if(masterproc) then
    call readnoahcrd(noahdrv)
endif
call def_noahpar_struct
call MPI_BCAST(noahdrv, 1, MPI_NOAHDRAV_STRUCT, 0, &
    MPI_COMM_WORLD, ierr)
#ifndef OPENDAP
    noahdrv%noah_albfile = trim(opendap_home)//trim(adjustl(ciam)) &
        //''//noahdrv%noah_albfile
    noahdrv%noah_mgfile = trim(opendap_home)//trim(adjustl(ciam)) &
        //''//noahdrv%noah_mgfile
    noahdrv%noah_mxsnal = trim(opendap_home)//trim(adjustl(ciam)) &
        //''//noahdrv%noah_mxsnal
    noahdrv%noah_tbot = trim(opendap_home)//trim(adjustl(ciam)) &
        //''//noahdrv%noah_tbot
#endif
if(masterproc) then
    allocate(noah(nch))
else
```

```

    allocate(noah(di_array(iam)))
  endif
end subroutine noah_varder_ini

```

1.38.12 noah_writerst.F90 (Source File: noah_writerst.F90)

This program writes restart files for NOAH. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```

1 Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA
              initial conditions if necessary. This is controlled with
              local variable NOAAIC. Normally set to 0 in this subroutine
              but set to 1 if want to use Dag's NOAA IC's. Changed output
              directory structure, and commented out if-then check so that
              directory is always made.
28 Apr 2002: Kristi Arsenault; Added NOAH LSM into LDAS
28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
              and put SMC, SH20, STC limit for GDAS and GEOS forcing.
14 Jun 2003: Sujay Kumar , Separated the write restart from the original
              code

```

RESTART FILE FORMAT(fortran sequential binary):

```

YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
TILE(NCH)%COL           !Grid Col of Tile
TILE(NCH)%ROW           !Grid Row of Tile
TILE(NCH)%FGRD          !Fraction of Grid covered by tile
TILE(NCH)%VEGT          !Vegetation Type of Tile
NOAH(NCH)%STATES        !Model States in Tile Space

```

INTERFACE:

```

subroutine noah_writerst()
!uses:
use lisdrv_module, only : lis,tile
use time_module
USE noah_varder      ! NOAH tile variables
use time_manager
use tile_spmdMod

```

CONTENTS:

```

if(masterproc) then
!-----
! Restart Writing (2 files are written = active and archive)

```

```

!-----
      if((lis%t%gmt.eq.(24-noahdrv%writeintn)) &
         .or.lis%t%endtime.eq.1)then
        allocate(tmptilen(lis%d%nch))
        open(40,file=noahdrv%noah_rfile,form='unformatted') !Active archive restart
        call timemgr_write_restart(40)
        write(40) lis%p%vclass,lis%d%lnc,lis%d%lnr,lis%d%nch !Veg class, no tiles
        write(40) noah%t1           !NOAH Skin Temperature (K)
        write(40) noah%cmc          !NOAH Canopy Water Content
        write(40) noah%snowh         !NOAH Actual Snow Depth
        write(40) noah%sneqv         !NOAH Water Equivalent Snow Depth
        do l=1,4
          do t=1,lis%d%nch
            tmptilen(t)=noah(t)%stc(1)
          enddo
          write(40) tmptilen !NOAH Soil Temperature (4 layers)
        enddo
        do l=1,4
          do t=1,lis%d%nch
            tmptilen(t)=noah(t)%smc(1)
          enddo
          write(40) tmptilen !NOAH Total Soil Moist. (4 layers)
        enddo
        do l=1,4
          do t=1,lis%d%nch
            tmptilen(t)=noah(t)%sh2o(1)
          enddo
          write(40) tmptilen !NOAH Liquid Soil Moist. (4 layers)
        enddo
        write(40) noah%ch           !NOAH Heat/Moisture Sfc Exchange Coef.
        write(40) noah%cm           !NOAH Momentum Sfc Exchange Coef.
        close(40)
        write(*,*)'Noah Active restart written: ',noahdrv%noah_rfile
        write(unit=temp,fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
          lis%t%da,lis%t%hr
        read(unit=temp,fmt='(10a1)') ftime
        do i=1,10
          if(ftime(i).eq.(' '))ftime(i)='0'
        enddo
        write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2,a6,i3,a1)') &
          '/EXP',lis%o%expcode,'/NOAH/',lis%t%yr, &
          '/',lis%t%yr,lis%t%mo, &
          lis%t%da,'/LIS.E',lis%o%expcode,'.'
        read(unit=temp,fmt='(80a1)') (fname(i),i=1,37)
        do i=1,73
          if(fname(i).eq.(' '))fname(i)='0'
        enddo

```

```

        write(unit=temp,fmt='(a9)')'mkdir -p '
        read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)
        write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)') &
          '/EXP',lis%o%expcode,'/NOAH/', &
          lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
        read(unit=temp,fmt='(80a1)') (fyrmodir(i),i=1,26)
        do i=1,26
          if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
        enddo

        write(unit=temp,fmt='(a8)')'.Noahrst'
        read(unit=temp,fmt='(80a1)') (fsubs(i),i=1,8)

        write(unit=temp,fmt='(a40)') lis%o%odir
        read(unit=temp,fmt='(80a1)') (fbase(i),i=1,80)
        c=0
        do i=1,80
          if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
        enddo
        write(unit=temp,fmt='(80a1)')(fbase(i),i=1,c),(fname(i),i=1,37), &
          (ftime(i),i=1,10),(fsubs(i),i=1,8)
        read(unit=temp,fmt='(a80)')filen

        write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
          (fyrmodir(i),i=1,26)
        read(unit=temp,fmt='(a80)')mkfyrm0

!-----
! Archive File Name Generation Complete
! Make the directories for the NOAH restart file
!-----
      CALL SYSTEM(MKFYRMO)
!-----
! Archive File Name Generation Complete
!-----
      open(40,file=filen,status='unknown',form='unformatted')
      write(40) lis%p%vclass,lis%d%lnc,lis%d%lnr,lis%d%nch !veg class, no tiles
      write(40) noah%t1           !noah skin temperature (k)
      write(40) noah%cmc         !noah canopy water content
      write(40) noah%snowh       !noah actual snow depth
      write(40) noah%sneqv       !noah water equivalent snow depth
      do l=1,4
        do t=1,lis%d%nch
          tmptilen(t)=noah(t)%stc(l)
        enddo
        write(40) tmptilen !noah soil temperature (4 layers)
      enddo
      do l=1,4

```

```

        do t=1,lis%d%nch
            tmptilen(t)=noah(t)%smc(1)
        enddo
        write(40) tmptilen !noah total soil moist. (4 layers)
    enddo
    do l=1,4
        do t=1,lis%d%nch
            tmptilen(t)=noah(t)%sh2o(l)
        enddo
        write(40) tmptilen !noah liquid soil moist. (4 layers)
    enddo
    write(40) noah%ch           !noah heat/moisture sfc exchange coef.
    write(40) noah%cm           !noah momentum sfc exchange coef.

    close(40)

    write(*,*)'noah archive restart written: ',filen
    deallocate(tmptilen)
endif
endif
return

```

1.38.13 readnoahcrd.F90 (Source File: readnoahcrd.F90)

Routine to read Noah specific parameters from the card file.

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Code

INTERFACE:

subroutine readnoahcrd(noahdrv)

USES:

use noahdrv_module

CONTENTS:

```

open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=noah)
print*, 'Running NOAH LSM:'
print*, 'NOAH Active Restart File: ', noahdrv%NOAH_RFILE
noahdrv%noah_gfractime = 0.0
noahdrv%noah_albtime = 0
noahdrv%noah_albdchk = 0
noahdrv%noah_gfracdchk = 0

```

```

noahdrv%NOAHOPEN=0
noahdrv%NOAH_ZST      = 9

close(11)

```

1.38.14 setnoahp.F90 (Source File: setnoahp.F90)

This subroutine retrieves NOAH parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

28 Apr 2002: Kristi Arsenault; Added NOAH LSM, Initial Code
 13 Oct 2003: Sujay Kumar; Domain independent modifications

INTERFACE:

subroutine setnoahp

USES:

```

use lisdrv_module, only : grid,tile,lis
use noah_varder      ! NOAH tile variables
#if ( defined OPENDAP )
  use opendap_module
#endif

```

CONTENTS:

```

#if ( ! defined OPENDAP )
  integer :: tnroffset = 0
#endif
!-----
! Convert UMD Classes to SIB Classes for Each Tile
!-----
print*, 'MSG: setnoahp -- Calling MAPVEGC to convert UMD to SIB', &
         (' , iam, ')
print*, 'DBG: setnoahp -- nch',lis%d%nch,' ( ,iam, )'

do n=1,lis%d%nch
  call mapvegc(tile(n)%vegt)
  noah(n)%vegt = tile(n)%vegt
enddo
print*, 'DBG: setnoahp -- left MAPVEGC', (' ,iam, ')
!-----
! Get Vegetation Parameters for NOAH Model in Tile Space
! Read in the NOAH Static Vegetation Parameter Files
!-----
```

```

open(unit=11,file=noahdrv%noah_vfile,status='old')

do j=1,noahdrv%noah_nvegp
    read(11,*)(value(i,j),i=1,lis%p%nt)
enddo
close(11)
!-----
! Assign STATIC vegetation parameters to each tile based on the
! type of vegetation present in that tile.
! These parameters will be stored in one long array--structured
! as follows: Tile 1, all the parameters (1 through numparam)
! then Tile 2, all the parameters
! Then Tile 3, all the parameters etc.
!-----

do i=1,lis%d%nch
    do j=1,noahdrv%noah_nvegp
        noah(i)%vegp(j)=value(tile(i)%vegt,j)
    enddo
enddo
!-----
! Read in bottom temperature fields and adjust for elevation differnce
! with either Eta (NLDAS) or GDAS (GLDAS) correction datasets.
!-----

#if ( defined OPENDAP )
    allocate(placetbot(parm_nc,1+nroffset:parm_nr+nroffset))
#else
    allocate(placetbot(lis%d%gnc,lis%d%gnr))
#endif

#if ( defined OPENDAP )
    print*, 'MSG: setnoahp -- Retrieving TBOT file ', &
        trim(noahdrv%noah_tbot), ' (,iam,)'
    call system("opendap_scripts/gettbtop.pl "//ciam// " // &
        trim(noahdrv%noah_tbot)//" // &
        cparm_slat// " //cparm_nlat// " // &
        cparm_wlon// " //cparm_elon)
#endif
print *, 'Opening GLDAS TBOT File ', noahdrv%NOAH_TBOT
OPEN(UNIT=12, FILE=noahdrv%NOAH_TBOT, &
    FORM="UNFORMATTED")

READ(12) PLACETBOT
CLOSE(12)

print*, 'MSG: setnoahp -- Read TBOT file', (',iam,)'
do i = 1, lis%d%nch
    if(placetbot(tile(i)%col,tile(i)%row-tnroffset).ne.-9999.00) then

```

```

        noah(i)%tempbot = placetbot(tile(i)%col, tile(i)%row-tnroffset)
        endif
    enddo
    deallocate(placetbot)
    call absoft_release_cache()

!-----
! The MAX SNOW ALBEDO field is opened and read in here:
!-----

#if ( defined OPENDAP )
    allocate(tmpalb(parm_nc,1+nroffset:parm_nr+nroffset))
#else
    allocate(tmpalb(lis%d%gnc,lis%d%gnr))
#endif

#if ( defined OPENDAP )
    print*, 'MSG: setnoahp -- Retrieving MAXSNALB file ', &
        trim(noahdrv%noah_mxsnal), ',(,iam,)'
    call system("opendap_scripts/getmaxsnalb.pl "//ciam//"/" //&
        trim(noahdrv%noah_mxsnal)//" //"
        & cparm_slat//" //cparm_nlat//" //"
        & cparm_wlon//" //cparm_elon)
#endif

print *, 'MSG: setnoahp -- Opening GLDAS MAXSNALB File', &
    trim(noahdrv%noah_mxsnal), ',(,iam,)'
OPEN(UNIT=12,FILE=noahdrv%NOAH_MXSNAL, &
    FORM="UNFORMATTED")

READ(12) TMPALB
CLOSE(12)

do i=1,lis%d%ncn
    if(tmpalb(tile(i)%col, tile(i)%row-tnroffset).ne.-9999.00) then
        noah(i)%mxsnalb = tmpalb(tile(i)%col, tile(i)%row-tnroffset)
    endif
enddo

deallocate(tmpalb)
call absoft_release_cache()

print*, 'MSG: setnoahp -- Read mxsnalb', ',(,iam,)'
print*, 'MSG: setnoahp -- reading soil and clay files', ',(,iam,)'
!-----
! Open soil files (sand, clay, and porosity).
!-----

#if ( defined OPENDAP )
    allocate(sand1(parm_nc,parm_nr))

```

```

allocate(clay1(parm_nc,parm_nr))
print*, 'MSG: setnoahp -- Retrieving SAND file ', &
trim(lis%p%safile), ',(,iam,)'
call system("opendap_scripts/getsand.pl "//ciam//" //&
trim(lis%p%safile)//" //&
cparm_slat//" //cparm_nlat//" //&
cparm_wlon//" //cparm_elon)
print*, 'MSG: setnoahp -- Retrieving CLAY file ', &
trim(lis%p%clfile), ',(,iam,)'
call system("opendap_scripts/getclay.pl "//ciam//" //&
trim(lis%p%clfile)//" //&
cparm_slat//" //cparm_nlat//" //&
cparm_wlon//" //cparm_elon)

#else
allocate(sand1(lis%d%gnc,lis%d%gnr))
allocate(clay1(lis%d%gnc,lis%d%gnr))
#endif
OPEN(15,FILE=LIS%P%SAFILE,FORM='UNFORMATTED',STATUS='OLD')
OPEN(16,FILE=LIS%P%CLFILE,FORM='UNFORMATTED',STATUS='OLD')
!-----
!      Read soil properties and convert to Zobler soil classes.
!      Note that the 3 files each contain 3 global records, one
!      for each layer depth (0-2, 2-150, 150-350 cm). At this
!      time only the top layer data are used to map soil
!      parameters. Since NOAH has 4 layers, the third layer of
!      porosity is temporarily used for layer 4 as well.
!-----
READ(15) SAND1
READ(16) CLAY1
CLOSE(15)
CLOSE(16)
print*, 'MSG: setnoahp -- read sand and clay files',(,iam,)'
!-----
! Determine Zobler-equivalent soil classes derived from
! percentages of sand and clay, used in NOAH.
!-----
#if ( defined OPENDAP )
allocate(soiltyp(parm_nc,1+nroffset:parm_nr+nroffset))
CALL SOILTYPE(parm_nc,parm_nr,SAND1,CLAY1,SOILTYP)
#else
allocate(soiltyp(lis%d%gnc,lis%d%gnr))
CALL SOILTYPE(LIS%D%GNC,LIS%D%GNR,SAND1,CLAY1,SOILTYP)
#endif

deallocate(sand1)
deallocate(clay1)
call absoft_release_cache()
!-----

```

```

! Read in the NOAH Soil Parameter File
!-----
OPEN(UNIT=18,FILE=noahdrv%NOAH_SFILE,STATUS='OLD', &
      ACCESS='SEQUENTIAL')

DO I=1,noahdrv%NOAH_NSOILP
  READ(18,*)(BASICSET(JJ,I),JJ=1,noahdrv%NOAH_ZST)
ENDDO
CLOSE(18)
print*, 'MSG: setnoahp -- read sfile ', trim(noahdrv%NOAH_SFILE), &
         (' , iam, ')
!-----
! Convert grid space to tile space for soil type values.
!-----
allocate(placesltyp(lis%d%nch))
do i=1,lis%d%nch
  placesltyp(i) = soiltyp(tile(i)%col, tile(i)%row-tnroffset)
  noah(i)%zobsoil = placesltyp(i)
end do

if(lis%o%wparam.eq.1) then
  allocate(stype(lis%d%lnc,lis%d%lnr))
  do i=1,lis%d%nch
    if(grid(i)%lat*1000.ge.lis%d%kgds(4).and. &
       grid(i)%lat*1000.le.lis%d%kgds(7).and. &
       grid(i)%lon*1000.ge.lis%d%kgds(5).and. &
       grid(i)%lon*1000.le.lis%d%kgds(8)) then
      rindex = tile(i)%row - (lis%d%kgds(4)-lis%d%kgds(44)) &
                 /lis%d%kgds(9)
      cindex = tile(i)%col - (lis%d%kgds(5)-lis%d%kgds(45)) &
                 /lis%d%kgds(10)
      stype(cindex,rindex) = noah(i)%zobsoil(1)*1.0
    endif
  enddo

  open(32,file="soiltype.bin",form='unformatted')
  write(32) stype
  close(32)
  deallocate(stype)
endif

deallocate(soiltyp)
call absoft_release_cache()
!-----
! Assign SOIL Parameters to each tile based on the
! type of Zobler soil class present in that tile.
!-----
DO I=1,lis%d%nch          !Tile loop

```

```

K=PLACESLtyp(I)           !Soil type
DO J=1,noahdrv%NOAH_NSOILP !Soil parameter loop
  NOAH(I)%SOILP(J)=BASICSET(K,J)
ENDDO !J
ENDDO !I
deallocate(placesltyp)
call absoft_release_cache()
return

```

1.38.15 sh2o_init.F90 (Source File: sh2o_init.F90)

To do a 'cold start' initialization of soil liquid water SH2O for NOAH LSM, though also adaptable for other land-surface models, using either GDAS or Eta forcing data.

REVISION HISTORY:

```

NCEP; Original code developed by NCEP for Eta model
      subroutines to initialize soil liquid water, SH2O
04 Nov 2002: Kristi Arsenault; Modified code to be used with noahrst.f
      to initialize NOAH with NCEP forcing data

```

INTERFACE:

```

subroutine sh2oinit(smc,stc,smcmax,psis,beta,sh2o)

implicit none

```

ARGUMENTS:

```

REAL STC      ! NOAH Soil Layer Temperature (K)
REAL SMC      ! NOAH Soil Layer Total Moisture (liq+frzn)
REAL SH2O     ! NOAH Soil Layer Liquid Moisture

REAL PSIS          ! Saturated soil potential
REAL BETA         ! B-parameter
REAL SMCMAX       ! Max soil moisture content (porosity)
REAL BX

```

CONTENTS:

```

! -----
! COLD START: determine liquid soil water content (SH2O)
! NSOIL number of soil layers
! -----
! SH2O <= SMC for T < 273.149K (-0.001C)
IF (STC .LT. 273.149) THEN
! -----
! first guess following explicit solution for Flerchinger Eqn from Koren

```

```

! et al, JGR, 1999, Eqn 17 (KCOUNT=0 in FUNCTION FRH20).
! -----
BX = BETA
IF ( BETA .GT. BLIM ) BX = BLIM
FK=((HLICE/(GRAV*(-PSIS)))* &
      ((STC-T0)/STC))**(-1/BX))*SMCMAX
IF (FK .LT. 0.02) FK = 0.02
SH20 = MIN ( FK, SMC )
! -----
! now use iterative solution for liquid soil water content using
! FUNCTION FRH20 with the initial guess for SH20 from above explicit
! first guess.
! -----
SH20 = FRH20(STC,SMC,SH20,SMCMAX,BETA,PSIS)

ELSE
! -----
! SH20 = SMC for T => 273.149K (-0.001C)
SH20=SMC
! -----
ENDIF

RETURN

```

1.38.16 opendap_init_c_struct (Source File: opendap_wrappers.F90)

This file contains routines that defines OPENDAP specific routines

REVISION HISTORY:

14 Oct 2003; James Geiger :Initial Specification

INTERFACE:

```
subroutine opendap_init_c_struct
```

USES:

```
#if ( defined OPENDAP )
  use spmdMod, only : iam
  use opendap_module, only : parm_nc, parm_nr,      &
    parm_slat, parm_nlat, &
    parm_wlon, parm_elon, &
    tnroffset
```

CONTENTS:

```
call setup_c_struct(iam,                      &
    parm_nc, parm_nr,      &
    parm_slat, parm_nlat, &
    parm_wlon, parm_elon, &
    tnroffset)
```

1.38.17 readviccrd.F90 (Source File: readviccrd.F90)

Routine to read Vic specific parameters from the card file.

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readviccrd(vicdrv)
```

USES:

```
use vicdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=vic)
PRINT*, 'Running VIC LSM:'
PRINT*, 'VIC Soil File: ', vicdrv%VIC_SFILE
vicdrv%vic_quick_flux = 1 !default
if(vicdrv%vic_full_energy==1) then
    vicdrv%vic_quick_flux = 1
    vicdrv%vic_grnd_flux = 1
endif
if(vicdrv%vic_frozen_soil==1) then
    vicdrv%vic_quick_flux = 0
    vicdrv%vic_grnd_flux = 1
endif
vicdrv%vicopen = 0
vicdrv%VIC_FULL_ENERGY = 1
vicdrv%VIC_FROZEN_SOIL = 0

close(11)
```

1.38.18 vic_atmdrv.F90 (Source File: vic_atmdrv.F90)

Transfer forcing from grid to vic tile space.

REVISION HISTORY:

14 Apr 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine vic_atmdrv(t, forcing)
```

USES:

```
use lisdrv_module , only : lis      ! LDAS non-model-specific 1-D variables
```

CONTENTS:

```
call vic_f2t(t, forcing,lis%t%ts)
return
```

1.39 Fortran: Module Interface vicdrv_module.F90 (Source File: vicdrv_module.F90)

Module for runtime specific VIC variables

REVISION HISTORY:

14 Oct 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module vicdrv_module
```

ARGUMENTS:

```
type vicdrvdec
    integer :: vic_nlayer      !Number of soil layers in VIC
    integer :: vic_nnode        !Number of soil thermal nodes in the model
    integer :: vic_snowband     !Number of snow bands
    integer :: vic_rootzones
    integer :: vic_full_energy !FLAG
    integer :: vic_grnd_flux   !GRND_FLUX
    integer :: vic_frozen_soil !FROZEN_SOIL
    integer :: vic_quick_flux  !quick_flux
    integer :: vicopen
    character*40:: vic_sfile   !VIC SOIL FILE
    character*40:: vic_veglibfile !VIC veg library
    character*40:: vic_rfile   !VIC restart file
    real :: writeintvic        !Counts number of output times for VIC
end type vicdrvdec
```

1.39.1 vic_dynsetup.F90 (Source File: vic_dynsetup.F90)

Updates the time dependent VIC variables

REVISION HISTORY:

14 Apr 2003: Sujay Kumar: Initial code

INTERFACE:

```
subroutine vic_dynsetup()
```

1.39.2 vic_main.F90 (Source File: vic_main.F90)

Initializes vic model state and calls the VIC physics routines

REVISION HISTORY:

14 Apr 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine vic_main()
```

USES:

```
use lisdrv_module, only: lis,tile
use vic_varder, only : vicdrv
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
outputflag = 0
call initialize_model_state(vicdrv%vic_nlayer,    &
    vicdrv%vic_snowband,vicdrv%vic_nnode,          &
    vicdrv%vic_quick_flux,vicdrv%vic_grnd_flux,   &
    vicdrv%vic_frozen_soil);
if(lis%t%tscount==0 .or. lis%t%tscount ==1) then
    outputflag = 1
endif
call vic_run(vicdrv%vic_nlayer,vicdrv%vic_nnode,      &
    vicdrv%vic_snowband,lis%t%da,lis%t%doy,lis%t%hr, &
    lis%t%mo,lis%t%yr,                                &
    vicdrv%vic_full_energy,vicdrv%vic_frozen_soil,     &
    vicdrv%vic_grnd_flux,vicdrv%vic_quick_flux, outputflag);
```

1.39.3 vic_output.F90 (Source File: vic_output.F90)

This subroutines sets up methods to write VIC output

REVISION HISTORY:

14 Apr 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine vic_output
```

USES:

```
use lisdrv_module, only : lis
use vic_varder, only : vicdrv
use spmdMod, only : masterproc,npes
```

CONTENTS:

```
if(lis%o%wsingle ==1) then
    if(mod(lis%t%gmt,vicdrv%writeintvic).eq.0) then
!-----
! Writes a separate output file for each variable
!-----
    do i=1,24
        call vic_singlegather(i)
        if ( masterproc ) then
            call vic_singleout(lis%o%startcode, vicdrv%vicopen,&
                lis%d%glbnch,lis%o%wfor,lis%o%wout, &
                lis%o%expcode,vicdrv%vic_snowband,vicdrv%vic_nlayer,&
                i)
        endif
    enddo
    do i=27,27
        call vic_singlegather(i)
        if ( masterproc ) then
            call vic_singleout(lis%o%startcode, vicdrv%vicopen,&
                lis%d%glbnch,lis%o%wfor,lis%o%wout, &
                lis%o%expcode,vicdrv%vic_snowband,vicdrv%vic_nlayer,&
                i)
        endif
    enddo
    call vic_totinit()
endif
else
!-----
! Write bundled output
!-----
    if(mod(lis%t%gmt,vicdrv%writeintvic).eq.0) then
        if(npes > 1) then
```

```

        call vic_gather()
    endif
    if(masterproc) then
        call vic_almaout(lis%o%startcode, vicdrv%vicopen, &
            lis%d%glbnch,lis%o%wfor, lis%o%wout, &
            lis%o%expcode,vicdrv%vic_snowband,vicdrv%vic_nlayer)
    endif
    call vic_totinit()
    endif
endif

```

1.40 Fortran: Module Interface vicpardef_module.F90 (Source File: vicpardef_module.F90)

This module contains routines that defines MPI derived data types for VIC LSM

REVISION HISTORY:

14 Oct 2003; Sujay Kumar Initial Specification

INTERFACE:

```
module vicpardef_module
```

USES:

```
use vicdrv_module
use spmdMod
```

1.40.1 def_vicpar_struct (Source File: vicpardef_module.F90)

Routine that defines MPI derived data types for VIC

INTERFACE:

```
subroutine def_vicpar_struct()
```

1.40.2 vic_readrestart.F90 (Source File: vic_readrestart.F90)

This program reads restart files for VIC.

REVISION HISTORY:

17 Nov 2003; Justin Sheffield : Initial Version

21 Nov 2003; Sujay Kumar : Added the time manager restart.

INTERFACE:

```
subroutine vic_readrestart
```

USES:

```
use lisdrv_module, only: lis,tile
use vic_varder, only : vicdrv
use time_manager
use time_module
implicit none
integer :: curSec
```

CONTENTS:

```
! if(lis%o%startcode .eq. 1) then
!   OPEN(40,FILE='time.rst',FORM='unformatted')
!   call timemgr_read_restart(40)
!   call timemgr_restart()
!   call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
!   call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
!   call updatetime(lis%t)
!   call read_initial_model_state(trim(vicdrv%VIC_RFILE), &
!                                 len(trim(vicdrv%VIC_RFILE)), &
!                                 vicdrv%vic_snowband, vicdrv%vic_nlayer, vicdrv%vic_nnode)
!   close(40)
! endif
```

1.40.3 vic_setup.F90 (Source File: vic_setup.F90)

Completes the setup routines for VIC

REVISION HISTORY:

21 Nov 2003; Sujay Kumar : Initial Specification

INTERFACE:

```
subroutine vic_setup()
```

USES:

```
use lisdrv_module, only: lis,tile
use vic_varder, only : vicdrv
use spmdMod, only : masterproc, npes,iam
#if ( defined OPENDAP )
  use opendap_module
#endif
```

CONTENTS:

```

#if ( defined OPENDAP )
  nc = lis%d%lnc
  nr = lis%d%lnr
#else
  nc = lis%d%gnc
  nr = lis%d%gnr
#endif
#if ( ! defined OPENDAP )
  if(masterproc) then
#endif
  t = len(trim(vicdrv%vic_veglfile))
  call read_veglib(trim(vicdrv%vic_veglfile),t, &
    ntype,vicdrv%vic_rootzones)
  t = len(trim(vicdrv%vic_sfile))
  t1 = len(trim(lis%p%safile))
  t2 = len(trim(lis%p%clfile))
  call read_soilparam(trim(vicdrv%vic_sfile),lis%d%nch, &
    t,vicdrv%vic_nlayer,vicdrv%vic_nnode,           &
    trim(lis%p%safile),t1,trim(lis%p%clfile),t2,   &
    nc,nr,tile%col,tile%row)
#endif
#if ( ! defined OPENDAP )
  endif
#endif

#if ( ! defined OPENDAP )
  call vic_bcast()
  if(npes > 1) call vic_scatter()
  print*, 'DBG: vic_setup -- Done scatter', (',iam,')
#endif
call read_vegparam(tile%vegt)
call calc_root_fractions(lis%d%nch, &
  vicdrv%vic_nlayer,vicdrv%vic_rootzones)
call make_dist_prcp(vicdrv%vic_snowband)
call vic_totinit()

call vic_soilparamalloc(vicdrv%vic_nlayer,vicdrv%vic_nnode, &
  vicdrv%vic_snowband);
print*, 'MSG: vic_setup -- Done', (',iam,')

```

1.41 Fortran: Module Interface vic_varder.F90 (Source File: vic_varder.F90)

Module for 1-D VIC land model driver initialization.

REVISION HISTORY:

21 Nov 2003; Sujay Kumar : Initial Specification

INTERFACE:

```
module vic_varder
```

USES:

```
use tile_spmdMod
use vicdrv_module
use vicpardef_module
```

1.41.1 vic_varder_ini (Source File: vic_varder.F90)

Reads in runtime VIC parameters, allocates memory for variables

INTERFACE:

```
subroutine vic_varder_ini(nch)
```

CONTENTS:

```
if(masterproc) then
    call readviccrd(vicdrv)
endif
call def_vicpar_struct
call MPI_BCAST(vicdrv, 1, MPI_VICDRV_STRUCT, 0, &
    MPI_COMM_WORLD, ier)
#if ( defined OPENDAP )
    call opendap_init_c_struct
#endif
call vic_allocate(nch, di_array,displs)
```

1.41.2 vic_writerst.F90 (Source File: vic_writterestart.F90)

This program writes restart files for VIC. This includes all relevant water/energy storages, tile information, and time information.

REVISION HISTORY:

17 Nov 2003; Justin Sheffield : Initial Version

21 Nov 2003; Sujay Kumar : Added the time manager restart.

INTERFACE:

```
subroutine vic_writterestart
```

USES:

```
use lisdrv_module, only: lis,tile
use vic_varder, only : vicdrv
use time_manager
```

CONTENTS:

```
! if((lis%t%gmt .eq. (24-vicdrv%writeintvic)) &
!     .or.lis%t%endtime .eq. 1) then
!   OPEN(40,FILE="time.rst",FORM='unformatted')
!   call timemgr_write_restart(40)
!   write(*,*) vicdrv%vic_snowband, vicdrv%vic_nlayer, vicdrv%vic_nnode
!   call write_model_state(trim(vicdrv%VIC_RFILE), &
!     len(trim(vicdrv%VIC_RFILE)), &
!     vicdrv%vic_snowband, vicdrv%vic_nlayer, vicdrv%vic_nnode)
!   close(40)
! endif
```

1.42 Fortran: Module Interface **cmapdomain_module.F90** (Source File: **cmapdomain_module.F90**)

Contains routines and variables that define the native domain for CMAP precipitation product.

INTERFACE:

```
module cmapdomain_module
```

USES:

```
use cmapdrv_module
```

ARGUMENTS:

```
type(cmapdrvdec) :: cmapdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:)
integer, allocatable :: n12(:)
integer, allocatable :: n21(:)
integer, allocatable :: n22(:)
real, allocatable :: w11(:,w12(:)
real, allocatable :: w21(:,w22(:)
```

1.42.1 defnatcmap.F90 (Source File: *cmapdomain_module.F90*)

Defines the kgds array describing the native forcing resolution for CMAP data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnatcmap()
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

ARGUMENTS:

```
integer :: kgdsi(200)
```

CONTENTS:

```
call readcmapcrd(cmapdrv)
kgdsi(1) = 4
kgdsi(2) = 512
kgdsi(3) = 256
kgdsi(4) = 89463
kgdsi(5) = 0
kgdsi(6) = 128
kgdsi(7) = -89463
kgdsi(8) = -703
kgdsi(9) = 703
kgdsi(10) = 128
kgdsi(20) = 255
call allocate_cmap_ip(lis%d%lnc*lis%d%lnr)
call def_cmap_ip_input(kgdsi)
```

1.42.2 allocate_cmap_ip (Source File: *cmapdomain_module.F90*)

Allocates memory for CMAP interpolation variables

INTERFACE:

```
subroutine allocate_cmap_ip(N)
```

CONTENTS:

```

allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n))
allocate(n12(n))
allocate(n21(n))
allocate(n22(n))
allocate(w11(n))
allocate(w12(n))
allocate(w21(n))
allocate(w22(n))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0

```

1.42.3 def_cmap_ip_input (Source File: *cmapdomain_module.F90*)

Calculates weights and neighbor information required for CMAP interpolation

INTERFACE:

```
subroutine def_cmap_ip_input (kgds)
```

USES:

```

use spmdMod
use lisdrv_module, only:lis

```

CONTENTS:

```

!-----
! Calls the routines to decode the grid description and
! calculates the weights and neighbor information to perform
! spatial interpolation. This routine eliminates the need to
! compute these weights repeatedly during interpolation.
!-----
      mo = lis%d%lnc*lis%d%lnr
      if(kgdso(1).ge.0) then
          call gdswiz(kgdso, 0,mo,fill,xpts,ypts,rlon,rlat,nn,0)
      endif
      call gdswiz(kgds,-1,nn,fill,xpts,ypts,rlon,rlat,nv,0)
      do n=1,nn
          xi=xpts(n)
          yi=ypts(n)
          if(xi.ne.fill.and.yi.ne.fill) then

```

```

i1=xi
i2=i1+1
j1=yi
j2=j1+1
xf=xi-i1
yf=yi-j1
n11(n)=ijkgds(i1,j1,kgds)
n21(n)=ijkgds(i2,j1,kgds)
n12(n)=ijkgds(i1,j2,kgds)
n22(n)=ijkgds(i2,j2,kgds)
if(min(n11(n),n21(n),n12(n),n22(n)).gt.0) then
    w11(n)=(1-xf)*(1-yf)
    w21(n)=xf*(1-yf)
    w12(n)=(1-xf)*yf
    w22(n)=xf*yf
else
    n11(n)=0
    n21(n)=0
    n12(n)=0
    n22(n)=0
endif
else
    n11(n)=0
    n21(n)=0
    n12(n)=0
    n22(n)=0
endif
enddo
mi = cmapdrv%ncold*cmapdrv%nrold
endif

```

1.43 Fortran: Module Interface *cmapdrv_module.F90* (Source File: *cmapdrv_module.F90*)

Module for runtime specific CMAP variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module cmapdrv_module
```

ARGUMENTS:

```

type cmapdrvdec
  integer :: ncold, nrold    !AWIPS 212 dimensions
  character*40 :: cmapdir    !CMAP Forcing Directory
  real*8 :: cmptime
end type cmapdrvdec

```

1.43.1 getcmap.F90 (Source File: getcmap.F90)

Opens and reads global precipitation forcing

CTIME = Current time

FTIMENRL = Nearest future data for NRL data

FTIMEHUFF = Nearest future data for HUFFMAN data

FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

```

17 Jul 2001: Jon Gottschalck; Initial code
10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
              using a ratio of the model convective / total ratio
30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources

```

INTERFACE:

```
subroutine getcmap()
```

USES:

```

use lisdrv_module, only : lis, gindex
use time_manager
use time_module, only : tick
use cmapdomain_module, only : cmapdrv
implicit none

```

CONTENTS:

```

endtime_cmap = 0
!-----
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-----
gap = 0.0001712328767098370
!-----
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-----
yr1 = lis%t%yr  !current time
mo1 = lis%t%mo
da1 = lis%t%da

```

```

hr1 = lis%t%hr
mn1 = lis%t%mn
ss1 = 0
ts1 = 0
call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
!-----
! CMAP product end time
!-----
yr5 = lis%t%yr !end accumulation time data
mo5 = lis%t%mo
da5 = lis%t%da
hr5 = 6*(lis%t%hr/6)
mn5 = 0
ss5 = 0
ts5 = 6*60*60
call tick( ftime_cmap, doy5, gmt5, yr5, mo5, da5, hr5, mn5, ss5, ts5 )
!-----
! Ensure that data is found during first time step
!-----
if ( lis%f%gpcpsrc.eq.4.and. get_nstep().eq. 1 ) endtime_cmap = 1
!-----
! Check for and get CMAP CPC Precipitation data
!-----
if (lis%f%gpcpsrc==4) then
  if ( ctime > cmapdrv%cmaptime ) endtime_cmap = 1
  if ( endtime_cmap == 1 ) then !get new time2 data
    ferror_cmap = 0
    call cmapfile( name, cmapdrv%cmapdir, yr5, mo5, da5, hr5 )
    print*, 'Getting new CMAP CPC precip data',name
    call glbp precip_cmap( name, ferror_cmap, hr5 )
    cmapdrv%cmaptime = ftime_cmap
  endif !need new time2
endif
return

```

1.43.2 cmapfile (Source File: getcmap.F90)

This subroutine puts together CMAP file name for 6 hour file intervals

INTERFACE:

```
subroutine cmapfile( name, cmapdir, yr, mo, da, hr)
```

CONTENTS:

```
91 format (a4)
92 format (80a1)
```

```
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a10)
98 format (a1, i4, i2, a1)
99 format (8a1)
!-----
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----
uyr = yr
umo = mo
uda = da
uhr = 6*(hr/6) !hour needs to be a multiple of 6 hours
umn = 0
uss = 0
ts1 = -24*60*60 !one day interval to roll back date.

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) cmapdir
read(90, 92, rec=1) (fbase(i), i=1,80)

write(90, 98, rec=1) '/', uyr, umo, '/'
read(90, 99, rec=1) fdir
do i = 1, 8
  if ( fdir(i) == ' ') fdir(i) = '0'
end do

write(90, 97, rec=1) 'cmap_gdas_'
read (90, 92, rec=1) (fsubs(i), i=1,10)

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
  if ( ftime(i) == ' ') ftime(i) = '0'
end do

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
  if ( ftime(i) == ' ') ftime(i) = '0'
end do

write(90, 91, rec=1) '.grb'
read (90, 92, rec=1) (fsubs2(i), i=1,4)
c = 0
do i = 1, 80
```

```

      if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
      end do

      write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,8),  &
                           (fsubs(i), i=1,10),(ftime(i), i=1,10),  &
                           (fsubs2(i), i=1,4)

      read(90, 93, rec=1) name

      close(90)
      return

```

1.43.3 *glbprecip_cmap.F90* (Source File: *glbprecip_cmap.F90*)

Includes reading routines for global CMAP precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

```

17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
              observations with domain 3 (2x2.5)
30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data

```

INTERFACE:

```
subroutine glbprecip_cmap( fname, ferror_cmap, filehr)
```

USES:

```

use lisdrv_module, only : lis, gindex
use obsprecipforcing_module, only: obsprecip
implicit none

```

ARGUMENTS:

```

character(len=80) :: fname           ! Filename variable for datafile
integer          :: ferror_cmap
integer          :: filehr

```

CONTENTS:

```

allocate (precip_regrid(lis%d%lnc,lis%d%lnr))
precip_regrid = -1.0
!-----
! Set necessary parameters for call to interp_gdas
!-----
ism      = 0
udef    = lis%d%udef

```

```

jj      = 0
if (mod((filehr),12).eq.0) then
  lugb=25
else
  lugb=26
endif
lugi    = 0
jpds    = -1
jpds(5) = 59
jpds(6) = 1
jpds(7) = 0
jgds    = 0
call baopen (lugb, fname, iret)
if (iret == 0 ) then
  call getgb (lugb,lugi,ncmap,jj,jpds,jgds,kf,k,kpds,kgdscmap,lb,cmapin,iret)
  call interp_cmap(kpds,kgdscmap,ncmap,cmapin,lb,lis%d%kgds, &
    lis%d%lnc,lis%d%lnr,precip_regrid)

do j = 1,lis%d%lnr
  do i = 1,lis%d%lnc
    if(precip_regrid(i,j) .eq. -1) then
      print*, j,i,precip_regrid(i,j)
    endif
    if (precip_regrid(i,j) .ne. -1.0) then
      index = gindex(i,j)
      if(index .ne. -1) then
        obsprecip(index) = precip_regrid(i,j)*3600.0
      endif
    endif
  enddo
enddo

call baclose (lugb,jret)

ferror_cmap = 1
close(10)
print*, "Obtained CMAP CPC precipitation data ", fname
else
  print*, "Missing CMAP CPC precipitation data ", fname
  ferror_cmap = 0
endif
deallocate (precip_regrid)

```

1.43.4 interp_cmap.F90 (Source File: interp_cmap.F90)

Interpolates CMAP observed precipitation forcing

INTERFACE:

```
subroutine interp_cmap(kpds,kgds,ngdas,f,lb,lis_gds,nc,nr, &
varfield)
```

USES:

```
use cmapdomain_module, only : mi,w11,w12,w21,w22,&
n11,n12,n21,n22,rlat,rlon

implicit none
```

ARGUMENTS:

```
integer :: nc, nr, ngdas, nglis
integer :: kpds(200),kgds(200), lis_gds(200)
real :: f(ngdas)
logical*1 :: lb(ngdas)
real, dimension(nc,nr) :: varfield
```

CONTENTS:

```
!-----
! Setting interpolation options (ip=0,bilinear)
! (km=1, one parameter, ibi=1,use undefined bitmap
! (needed for soil moisture and temperature only)
! Use budget bilinear (ip=3) for precip forcing fields
!-----

nglis = nc*nr
if (kpds(5)==59 .or. kpds(5)==214) then
    ip=3
    ipopt(1)=-1
    ipopt(2)=-1
    km=1
    ibi=1
else
    ip=0
    do i=1,20
        ipopt(i)=0
    enddo
    km=1
    ibi=1
endif
!-----
! Initialize output bitmap. Important for soil moisture and temp.
!-----
lo = .true.

! call ipolates (ip,ipopt,kgds,lis_gds,ngdas,nglis, &
!      km,ibi,lb,f,no,rlat,rlon,ibo,lo,lis1d,iret)
```

```

mi = ngdas
call polates0 (lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,&
    rlat, rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)
!-----
! Create 2D array for main program. Also define a "soil" mask
! due to different geography between GDAS & LDAS. For LDAS land
! points not included in GDAS geography dataset only.
!-----
count = 0
do j = 1, nr
    do i = 1, nc
        varfield(i,j) = lis1d(i+count)
        geogmask(i,j) = lo(i+count)
    enddo
    count = count + nc
enddo
!-----
! Save air tempertaure interpolated field for later use in
! initialization of soil temp where geography differs
! between GDAS and LDAS
!-----
if (kpds(5) .eq. 11 .and. kpds(6) .eq. 105) then
    do i = 1, nc
        do j = 1, nr
            geogtemp(i,j) = varfield(i,j)
        enddo
    enddo
endif

```

1.43.5 readcmapcrd.F90 (Source File: readcmapcrd.F90)

Routine to read CMAP specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readcmapcrd(cmapdrv)
```

USES:

```
use cmapdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
```

```

read(unit=11,NML=cmap)
print*, 'Using CMAP forcing'
print*, 'CMAP forcing directory : ', cmapdrv%CMAPDIR
!-----
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----
cmapdrv%cmptime = 0.0
close(11)

```

1.43.6 time_interp_cmap.F90 (Source File: time_interp_cmap.F90)

Calls the post processing utilities for handling observed precipitation data for CMAP

INTERFACE:

```
subroutine time_interp_cmap
```

USES:

```

use lisdrv_module, only:lis, grid
use obsprecipforcing_module, only : obsprecip
use grid_spmdMod

```

CONTENTS:

```

!-----
! Compute ratio between convective model precip and total model precip
! so that it can be applied to the observed global precip
!-----
do c = 1,gdi(iam)
    if (grid(c)%forcing(8) .ne. 0.0 .and. &
        grid(c)%forcing(8) .ne. lis%d%udef .and. &
        grid(c)%forcing(9) .ne. lis%d%udef) then
        ratio(c) = grid(c)%forcing(9) / grid(c)%forcing(8)
        if (ratio(c) .gt. 1.0) ratio(c) = 1.0
        if (ratio(c) .lt. 0.0) ratio(c) = 0.0
    else
        ratio(c) = 0.0
    endif
enddo
do c = 1, gdi(iam)
    if (obsprecip(c) .ne. -1.0) then
        grid(c)%forcing(8) = obsprecip(c) / 3600.0
        grid(c)%forcing(9) = ratio(c) * grid(c)%forcing(8)
    endif
enddo

```

1.43.7 gethuff.F90 (Source File: gethuff.F90)

Opens and reads global precipitation forcing

CTIME = Current time

FTIMENRL = Nearest future data for NRL data

FTIMEHUFF = Nearest future data for HUFFMAN data

FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

17 Jul 2001: Jon Gottschalck; Initial code

10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
using a ratio of the model convective / total ratio

30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources

INTERFACE:

```
subroutine gethuff
```

USES:

```
use lisdrv_module, only : lis, gindex
use time_manager
use time_module, only : tick,time2date
use huffdomain_module, only : huffdrv
implicit none
```

CONTENTS:

```
!-----
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-----
gap = 0.0001712328767098370
!-----
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-----
yr1 = lis%t%yr !current time
mo1 = lis%t%mo
da1 = lis%t%da
hr1 = lis%t%hr
mn1 = lis%t%mn
ss1 = 0
ts1 = 0
call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

!-----
! HUFFMAN product end time
!-----
```

```

yr3 = lis%t%yr !end accumulation time data
mo3 = lis%t%mo
da3 = lis%t%da
hr3 = 3*(lis%t%hr/3)
mn3 = 0
ss3 = 0
ts3 = 3*60*60
call tick( ftime_huff, doy3, gmt3, yr3, mo3, da3, hr3, mn3, ss3, ts3 )
breaktime = ftime_huff - ctime
datatime = ftime_huff
fnametime = ftime_huff
if (lis%f%gpcsrc == 3) then
  if (breaktime .ge. gap) then
    call time2date( datatime, kdoy3, kgmt3, kyr3, &
      kmo3, kda3, khr3, kmn3 )
    call time2date( fnametime, mdoy3, mgmt3, myr3, &
      mmo3, mda3, mhr3, mmn3 )
    flag1 = 1
    if (khr3 == 24) khr3 = 0
    if (mhr3 == 24) mhr3 = 0
    if (kgmt3 .eq. 0.0 .and. flag2 .eq. 2) then
      kts3 = -25.5*60*60
      call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
        kda3, khr3, kmn3, kss3, kts3 )
      mts3 = -27*60*60
      call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
        mda3, mhr3, mmn3, mss3, mts3 )
    else
      kts3 = -1.5*60*60
      call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
        kda3, khr3, kmn3, kss3, kts3 )
      mts3 = -3*60*60
      call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
        mda3, mhr3, mmn3, mss3, mts3 )
    endif
    flag2 = 1
  else
    if (get_nstep().eq. 1) then
      call time2date( datatime, kdoy3, kgmt3, kyr3, kmo3, &
        kda3, khr3, kmn3 )
      call time2date( fnametime, mdoy3, mgmt3, myr3, mmo3, &
        mda3, mhr3, mmn3 )
      if (kgmt3 .eq. 0) then
        mts3 = -24*60*60
        call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
          mda3, mhr3, mmn3, mss3, mts3 )
        kts3 = -22.5*60*60
        call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
          kda3, khr3, kmn3, kss3, kts3 )
      else
        mts3 = -27*60*60
        call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
          mda3, mhr3, mmn3, mss3, mts3 )
        kts3 = -3*60*60
        call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
          kda3, khr3, kmn3, kss3, kts3 )
      endif
    endif
  endif
endif

```

```

        kda3, khr3, kmn3, kss3, kts3 )
else
  mts3 = 0
  call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
             mda3, mhr3, mmn3, mss3, mts3 )
  kts3 = 1.5*60*60
  call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
             kda3, khr3, kmn3, kss3, kts3 )
endif
else
  flag1 = 2
  if (flag2 .eq. 1) then
    mts3 = 3*60*60
    call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
               mda3, mhr3, mmn3, mss3, mts3 )
    kts3 = 3*60*60
    call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
               kda3, khr3, kmn3, kss3, kts3 )
  endif
  flag2 = 2
endif
endif
endif
!-----
! Ensure that data is found during first time step
!-----
if ( lis%f%gpcpsrc.eq.3 .and. get_nstep().eq. 1 ) endtime_huff = 1
!-----
! Check for and get HUFFMAN observed Precipitation data
!-----
if (lis%f%gpcpsrc.eq.3) then
  if ( ctime > huffdrv%hufftime ) then
    endtime_huff = 1
    if ( endtime_huff == 1 ) then !get new time2 data
      print*, 'Getting new HUFFMAN satellite precip data', endtime_huff
      ferror_huff = 0
      call hufffile( name, huffdrv%huffdir, myr3, mmo3, mda3, mhr3 )
      call glbprecip_huff( name, ferror_huff )
      huffdrv%hufftime = datatime
    endif
  endif
endif
return

```

1.43.8 hufffile (Source File: gethuff.F90)

This subroutine puts together HUFFMAN file name for 3 hour file intervals

INTERFACE:

```
subroutine hufffile( name, huffdir, yr, mo, da, hr)
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a4)
98 format (a1, i4, i2, a1)
99 format (8a1)
89 format (a7)
!-----
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----

uyr = yr
umo = mo
uda = da
uhr = 3*(hr/3) !hour needs to be a multiple of 3 hours
umn = 0
uss = 0
ts1 = -24*60*60 !one day interval to roll back date.

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) huffdir
read(90, 92, rec=1) (fbase(i), i=1,80)

write(90, 98, rec=1) '/', uyr, umo, '/'
read(90, 99, rec=1) fdir
do i = 1, 8
  if ( fdir(i) == ' ') fdir(i) = '0'
end do

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
  if ( ftime(i) == ' ') ftime(i) = '0'
end do

write(90, 97, rec=1) '.bin'
read (90, 92, rec=1) (fsubs(i), i=1,4)
```

```

write(90, 89, rec=1) '3B42RT.'
read (90, 92, rec=1) (fprefix(i), i=1,7)
c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(90, 92, rec=1) (fbase(i),i=1,c),(fdir(i),i=1,8),(fprefix(i), i=1,7),  &
  (ftime(i), i=1,10), (fsubs(i), i=1,4)

read(90, 93, rec=1) name

close(90)
return

```

1.43.9 glbprecip_huff.F90 (Source File: glbprecip_huff.F90)

Includes reading routines for global HUFFMAN precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

```

17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
              observations with domain 3 (2x2.5)
30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data

```

INTERFACE:

```
subroutine glbprecip_huff (name_huff, ferror_huff )
```

USES:

```

use lisdrv_module, only : gindex
use obsprecipforcing_module, only: obsprecip

```

CONTENTS:

```

fname = name_huff
!-----
! Fill necessary arrays to assure not using old HUFFMAN data
!-----
precip = -1.0
obsprecip = -1.0
!-----
! Find HUFFMAN precip data, read it in and assign to forcing precip array.

```

```
! Must reverse grid in latitude dimension to be consistent with LDAS grid
!-----
open(unit=10,file= fname, status='old', &
&           access='direct',recl=xd*yd*4, &
&           form='unformatted',iostat=ios)

if (ios .eq. 0) then
  read (10,rec=1) head (1:2880),&
  ( ( rr (i, j), i = 1, xd ), j = 1, yd)
do i = 1,yd
  do j = 1,xd
    if (j .lt. 721) then
      precip(j,i) = float(rr(j+720,yd+1-i)) / 100.0
      if ( rr(j+720,yd+1-i) .eq. ibad ) precip(j,i) = -1.0
      if ( rr(j+720,yd+1-i) .lt. 0.0 ) precip(j,i) = -1.0
    else
      precip(j,i) = float(rr(j-720,yd+1-i)) / 100.0
      if ( rr(j-720,yd+1-i) .eq. ibad ) precip(j,i) = -1.0
      if ( rr(j-720,yd+1-i) .lt. 0.0 ) precip(j,i) = -1.0
    endif
  enddo
enddo
!-----
! Interpolating to desired domain and resolution
! Global precip datasets not used currently to force NLDAS
!-----
do i = 1,yd
  do j = 1,xd
    index = gindex(j,i)
    if (index .ne. -1) then
      obsprecip(index) = precip(j,i)
    endif
  enddo
enddo

ferror_huff = 1
close(10)
print*, "Obtained HUFFMAN precipitation data ", fname
else
  print*, "Missing HUFFMAN precipitation data ", fname
  ferror_huff = 0
endif
```

1.44 Fortran: Module Interface huffdomain_module.F90 (Source File: huffdomain_module.F90)

Contains routines and variables that define the native domain for HUFF precipitation product.

INTERFACE:

```
module huffdomain_module
```

USES:

```
use huffdrv_module
```

ARGUMENTS:

```
type(huffdrvdec) :: huffdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:)
integer, allocatable :: n12(:)
integer, allocatable :: n21(:)
integer, allocatable :: n22(:)
real, allocatable :: w11(:),w12(:)
real, allocatable :: w21(:),w22(:)
```

1.44.1 defnathuff.F90 (Source File: huffdomain_module.F90)

Defines the kgds array describing the native forcing resolution for HUFF data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnathuff()
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

ARGUMENTS:

```
integer :: kgdsi(200)
```

CONTENTS:

```
call readhuffcrd(huffdrv)
kgdsi = 0
call allocate_huff_ip(lis%d%lnc*lis%d%lnr)
```

1.44.2 allocate_huff_ip (Source File: huffdomain_module.F90)

Allocates memory for HUFF interpolation variables

INTERFACE:

```
subroutine allocate_huff_ip(N)
```

CONTENTS:

```
allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n))
allocate(n12(n))
allocate(n21(n))
allocate(n22(n))
allocate(w11(n))
allocate(w12(n))
allocate(w21(n))
allocate(w22(n))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.45 Fortran: Module Interface huffdrv_module.F90 (Source File: huffdrv_module.F90)

Module for runtime specific HUFF variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module huffdrv_module
```

ARGUMENTS:

```
type huffdrvdec
  integer :: ncold, nrld      !AWIPS 212 dimensions
  character*40 :: huffdir    !HUFF Forcing Directory
  real*8 :: hufftime
end type huffdrvdec
```

1.45.1 readhuffcrd.F90 (Source File: readhuffcard.F90)

Routine to read HUFF specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readhuffcrd(huffdrv)
```

USES:

```
use huffdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=huff)
print*, 'Using HUFF forcing'
print*, 'HUFF forcing directory :',huffdrv%HUFFDIR
!-----
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----
huffdrv%hufftime = 0.0
close(11)
```

1.45.2 time_interp_huff.F90 (Source File: time_interp_huff.F90)

Calls the post processing utilities for handling observed precipitation data for HUFF

INTERFACE:

```
subroutine time_interp_huff
```

CONTENTS:

! To be implemented

1.45.3 getpers.F90 (Source File: getpers.F90)

Opens and reads global precipitation forcing

CTIME = Current time

FTIMENRL = Nearest future data for NRL data

FTIMEHUFF = Nearest future data for HUFFMAN data

FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

```

17 Jul 2001: Jon Gottschalck; Initial code
10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
              using a ratio of the model convective / total ratio
30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources

```

INTERFACE:

```
subroutine getpers
```

USES:

```

use lisdrv_module, only : lis, gindex
use time_manager
use time_module, only : tick
use persdomain_module, only : persdrv
implicit none

```

CONTENTS:

```

!-----
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-----
gap = 0.0001712328767098370
!-----
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-----
yr1 = lis%t%yr !current time
mo1 = lis%t%mo
da1 = lis%t%da
hr1 = lis%t%hr
mn1 = lis%t%mn
ss1 = 0
ts1 = 0
call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

!-----
! PERSIANN product end time
!-----
yr4 = lis%t%yr !end accumulation time data
mo4 = lis%t%mo
da4 = lis%t%da
hr4 = 1*(lis%t%hr/1)
mn4 = 0
ss4 = 0
ts4 = 1*60*60
call tick( ftime_pers, doy4, gmt4, yr4, mo4, da4, hr4, mn4, ss4, ts4 )
!-----
```

```

! Ensure that data is found during first time step
!-----
if ( lis%f%gpcpsrc.eq.2 .and. get_nstep().eq. 1 ) endtime_pers = 1
!-----
! Check for and get Persiann Precipitation data
!-----
if (lis%f%gpcpsrc.eq.2) then
  if ( ctime > persdrv%perstime ) endtime_pers = 1
  if ( endtime_pers == 1 ) then !get new time2 data
    ferror_pers = 0
    call persfile( name, persdrv%persdir, yr4, mo4, da4, hr4 )
    print*, 'Getting new PERSIANN precip data',name
    call glbp precip_pers( name, ferror_pers )
    persdrv%perstime = ftime_pers
  endif !need new time2
endif

return

```

1.45.4 persfile (Source File: getpers.F90)

This subroutine puts together PERSIANN file name for 1 hour file intervals

INTERFACE:

```
subroutine persfile( name, persdir, yr, mo, da, hr)
```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a8)
98 format (a1, i4, i2, a1)
99 format (8a1)
89 format (a7)
!-----
! Make variables for the time used to create the file
! We don't want these variables being passed ou
!-----
uyr = yr
umo = mo
uda = da
uhr = 1*(hr/1) !hour needs to be a multiple of 3 hours
umn = 0

```

```

uss = 0
ts1 = -24*60*60 !one day interval to roll back date.

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) persdir
read(90, 92, rec=1) (fbase(i), i=1,80)

write(90, 98, rec=1) ' /', uyr, umo, ' /'
read(90, 99, rec=1) fdir
do i = 1, 8
  if ( fdir(i) == ' ') fdir(i) = '0'
end do

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
  if ( ftime(i) == ' ') ftime(i) = '0'
end do

write(90, 97, rec=1) '.lr_budi'
read (90, 92, rec=1) (fsubs(i), i=1,8)
c = 0
do i = 1, 80
  if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(90, 92, rec=1) (fbase(i),i=1,c),(fdir(i),i=1,8),  &
                   (ftime(i), i=1,10), (fsubs(i), i=1,8)

read(90, 93, rec=1) name

close(90)

return

```

1.45.5 glbpref_pers.F90 (Source File: glbpref_pers.F90)

Includes reading routines for global PERSIANN precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

17 Jul 2001: Jon Gottschalck; Initial code
 04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip observations with domain 3 (2x2.5)
 30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data

INTERFACE:

```
subroutine glbprecip_pers ( ld, gindex, name_pers, ferror_pers )
```

USES:

```
use lis_module
use obsprecipforcing_module, only: obsprecip
```

CONTENTS:

```
fname = name_pers
obsprecip = -1.0
!-----
! Determine offset in number of rows from 60 S
! since PERSIANN starts at 50 S
!-----
open(unit=10,file=fname, status='old',access='direct', &
      form='unformatted',recl=xd*yd*4,iostat=ios)

if (ios .eq. 0) then
  read (10,rec=1) precip

  do i = 1,yd
    do j = 1,xd
      index = gindex(j,i)
      col(index) = j
      row(index) = i
      if (index .ne. -1) then
        obsprecip(index) = precip(j,i)
      endif
    enddo
  enddo
  ferror_pers = 1
  close(10)
  print*, "Obtained PERSIANN precipitation data ", fname
else
  print*, "Missing PERSIANN precipitation data ", fname
  ferror_pers = 0
endif
```

1.46 Fortran: Module Interface persdomain_module.F90 (Source File: persdomain_module.F90)

Contains routines and variables that define the native domain for PERS precipitation product.

INTERFACE:

```
module persdomain_module
```

USES:

```
use persdrv_module
```

ARGUMENTS:

```
type(persdrvdec) :: persdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:)
integer, allocatable :: n12(:)
integer, allocatable :: n21(:)
integer, allocatable :: n22(:)
real, allocatable :: w11(:),w12(:)
real, allocatable :: w21(:),w22(:)
```

1.46.1 defnatpers.F90 (Source File: persdomain_module.F90)

Defines the kgds array describing the native forcing resolution for PERS data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnatpers()
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

ARGUMENTS:

```
integer :: kgdsi(200)
```

CONTENTS:

```
call readperscrd(persdrv)
kgdsi = 0
call allocate_pers_ip(lis%d%lnc*lis%d%lnr)
```

1.46.2 allocate_pers_ip (Source File: persdomain_module.F90)

Allocates memory for PERS interpolation variables

INTERFACE:

```
subroutine allocate_pers_ip(N)
```

CONTENTS:

```
allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n))
allocate(n12(n))
allocate(n21(n))
allocate(n22(n))
allocate(w11(n))
allocate(w12(n))
allocate(w21(n))
allocate(w22(n))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.47 Fortran: Module Interface persdrv_module.F90 (Source File: persdrv_module.F90)

Module for runtime specific PERS variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module persdrv_module
```

ARGUMENTS:

```
type persdrvdec
  integer :: ncold, nrld      !AWIPS 212 dimensions
  character*40 :: persdir    !PERS Forcing Directory
  real*8 :: perstime
end type persdrvdec
```

1.47.1 readperscrd.F90 (Source File: readperscard.F90)

Routine to read PERS specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readperscrd(persdrv)
```

USES:

```
use persdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=pers)
print*, 'Using PERS forcing'
print*, 'PERS forcing directory :',persdrv%PERSDIR
!-----
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----
persdrv%perstime = 0.0
close(11)
```

1.47.2 time_interp_pers.F90 (Source File: time_interp_pers.F90)

Calls the post processing utilities for handling observed precipitation data for PERS

INTERFACE:

```
subroutine time_interp_pers
```

CONTENTS:

```
! To be implemented
```

1.47.3 agrlwdn.F90 (Source File: agrlwdn.F90)

to compute the net downward longwave radiation at the earth's surface.

method:

=====

- calculate the emissivity of the clear sky.
- calculate downwelling longwave radiation of the clear sky.

- add contribution of low, middle and high clouds to the clear sky portion.

process narrative: flux3 - located in the flux3 sdf in dnxm

references:

dr idso's paper in the j. of geophys. research,
no 74, pp 5397-5403.

dr r.f.wachtmann's paper in the digest of preprints, topical meeting on remote sensing of the atmosphere, anaheim,ca, optical society of america, entitled, "expansion of atmospheric temperature-moisture profiles in empirical orthogonal functions for remote sensing applications", 1975

INTERFACE:

```
subroutine agrlwdn( sfctmp, e, iclamt, rldown )
```

REVISION HISTORY:

```
15 may 1988 initial version.....capt rice/sddc
07 sep 1999 ported to ibm sp-2. added intent attributes to
            arguments. updated prolog.....mr gayno/dnxm
25 oct 2001 implement in LDAS.....jesse meng/ncep
```

implicit none

INPUT PARAMETERS:

```
real,      intent(in)      :: iclamt  ( 3 )
real,      intent(in)      :: e
real,      intent(in)      :: sfctmp
```

OUTPUT PARAMETERS:

```
real,      intent(out)     :: rldown
```

CONTENTS:

```
!
!-----  

! executable code starts here...compute the cloud amount  

! in fraction of overcast (.1 to 1.0).  

!-----  

  

cldfrt(1) = iclamt(1) / 100.0  

cldfrt(2) = iclamt(2) / 100.0  

cldfrt(3) = iclamt(3) / 100.0  

  

!
!-----  

! convert vapor pressure units from pascals to millibars for use
```

```

!     in determining emissivities.
!
-----
```

$$\text{emb} = e * 0.01$$

```

!-----
```

$$\text{emissa} = 0.700 + (5.95e-5 * \text{emb} * \exp(1500 / \text{sfctmp}))$$

```

!-----
```

$$\text{use emissa in wachtmann's model for sky irradiance to calc a}$$

$$\text{resultant longwave downward radiation value. first calc a sasc}$$

$$\text{emmissivity (emissb), which is an adjusted idso emmisivity.}$$

$$\text{then use emissb to calculate the blackbody irradiance of the}$$

$$\text{clear sky (the 1st term of wachtmann's equation).}$$
!

$$\text{emissb} = -0.792 + (3.161 * \text{emissa}) - (1.573 * \text{emissa} * \text{emissa})$$

$$\text{clrsky} = \text{emissb} * \sigma * (\text{sfctmp} * \text{sfctmp} * \text{sfctmp} * \text{sfctmp})$$

```

!-----
```

$$\text{now compute the irradiance contribution from the low, middle,}$$

$$\text{and hi cloud layers (the 2nd thru 4th terms in wachtmann' eqn).}$$
!

$$\text{lcterm} = (80.0 - (5.0 * zl)) * \text{cldfrt}(1)$$

$$\text{mcterm} = (80.0 - (5.0 * zm)) * (1.0 - \text{cldfrt}(1)) * \text{cldfrt}(2)$$

$$\text{hcterm} = (80.0 - (5.0 * zh)) * (1.0 - \text{cldfrt}(1)) * &$$

$$& (1.0 - \text{cldfrt}(2)) * \text{cldfrt}(3)$$

```

!-----
```

$$\text{put it all together to get a resultant downwrd longwave irrad.}$$
!

$$\text{rldown} = \text{clrsky} + \text{hcterm} + \text{mcterm} + \text{lcterm}$$

```

return
```

1.48 Fortran: Module Interface agrmetdomain_module.F90 (Source File: agrmetdomain_module.F90)

Contains routines and variables that define theb native domain for AGRMET observed radiation forcing

INTERFACE:

```
module agrmetdomain_module
```

USES:

```
use agrmetdrv_module
```

1.48.1 defnatagrmet.F90 (Source File: agrmetdomain_module.F90)

Defines the kgds array describing the native forcing resolution for AGRMET data.

REVISION HISTORY:

11Dec2003: Sujay Kumar; Initial Specification

INTERFACE:

```
subroutine defnatagrmet()
```

USES:

```
use lisdrv_module, only :lis
implicit none
```

ARGUMENTS:

```
integer :: kgdsi(200)
```

CONTENTS:

```
call readagrmetcrd(agrmtdrv)
kgdsi = 0
kgdsi(1) = 0
kgdsi(2) = 1440
kgdsi(3) = 600
kgdsi(4) = -59875
kgdsi(5) = -179875
kgdsi(6) = 128
kgdsi(7) = 89500
kgdsi(8) = 179500
kgdsi(9) = 250
kgdsi(10) = 250
kgdsi(11) = 64
kgdsi(20) = 255
call allocate_agr_ip(lis%d%lnc*lis%d%lnr)
call def_agr_ip_input(kgdsi)
```

1.48.2 allocate_agr_ip (Source File: agrmetdomain_module.F90)

Allocate memory for AGRMET interpolation variables

INTERFACE:

```
subroutine allocate_agr_ip(N)
```

CONTENTS:

```
allocate(rlat(N))
allocate(rlon(N))
allocate(N11(N))
allocate(N12(N))
allocate(N21(N))
allocate(N22(N))
allocate(w11(N))
allocate(w12(N))
allocate(w21(N))
allocate(w22(N))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.48.3 def_agr_ip_input (Source File: agrmetdomain_module.F90)

Calculates weights and neighbor information required for AGRMET interpolation

INTERFACE:

```
subroutine def_agr_ip_input (kgds)
```

USES:

```
use spmdMod
use lisdrv_module, only:lis
```

CONTENTS:

```
!-----
! Calls the routines to decode the grid description and
! calculates the weights and neighbor information to perform
! spatial interpolation. This routine eliminates the need to
! compute these weights repeatedly during interpolation.
!-----
if(masterproc) then
```

```

kgdso = lis%d%kgds
mo = lis%d%lnc*lis%d%lnr
if(kgdso(1).ge.0) then
  call gdswiz(kgdso, 0,mo,fill,xpts,ypts,rlon,rlat,nn,0)
endif

call gdswiz(kgds,-1,nn,fill,xpts,ypts,rlon,rlat,nv,0)
do n=1,nn
  xi=xpts(n)
  yi=ypts(n)
  if(xi.ne.fill.and.yi.ne.fill) then
    i1=xi
    i2=i1+1
    j1=yi
    j2=j1+1
    xf=xi-i1
    yf=yi-j1
    n11(n)=ijkgs(i1,j1,kgds)
    n21(n)=ijkgs(i2,j1,kgds)
    n12(n)=ijkgs(i1,j2,kgds)
    n22(n)=ijkgs(i2,j2,kgds)
    if(min(n11(n),n21(n),n12(n),n22(n)).gt.0) then
      w11(n)=(1-xf)*(1-yf)
      w21(n)=xf*(1-yf)
      w12(n)=(1-xf)*yf
      w22(n)=xf*yf
    else
      n11(n)=0
      n21(n)=0
      n12(n)=0
      n22(n)=0
    endif
  else
    n11(n)=0
    n21(n)=0
    n12(n)=0
    n22(n)=0
  endif
enddo
mi = 864000
endif

```

1.49 Fortran: Module Interface agrmetdrv_module.F90 (Source File: agrmetdrv_module.F90)

Module containing runtime specific AGRMET variables

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Version

INTERFACE:

```
module agrmetdrv_module
```

ARGUMENTS:

```
type agrmetdrvdec
  integer :: ncold, nrold !AWIPS 212 dimensions
  integer :: nmif
  character*40 :: agrmetdir !AGRMET Forcing Directory
  real*8 :: agrmtime1, agrmtime2
end type agrmetdrvdec
```

1.49.1 getgrad.F90 (Source File: getgrad.F90)

Opens, reads, interpolates and overlays radiation forcing.

TIME1 = most recent past data TIME2 = most recent future data

REVISION HISTORY:

```
28 Oct 1999: Brian Cosgrove; Initial code, see getrad.f
27 Apr 2000: Brian Cosgrove' Disabled zenith angle correction cutoff for
             cos(zen) less than .2
11 May 2000: Brian Cosgrove; Enabled correction cutoffs for cos(zen) less
             than .1, stop model if computed value is greater than 1367 w/m2
08 Jan 2001: Brian Cosgrove; Added check to see if czb or czm is equal
             to zero before trying to divide by czm or czb. If it is
             zero, set radiation value to zero
06 Mar 2001: Brian Cosgrove; Changed computation of WT1 and WT2 in cases
             where the previous hour or the next hour of observed radiation
             is not available. Substituted TIME1 for LDAS%PINKTIME1 and
             LDAS%NESTIME1 and TIME2 for LDAS%PINKTIME2 and LDAS%NESTIME2
15 Jun 2001: Urszula Jambor; Reworked algorithm for AGRMET data & GLDAS.
15 Oct 2001: Jesse Meng; Replace lis%agrmet flag by lis%agrmetsw and
             lis%agrmetlw; added call retagrlw() to calculate
             AGRMET LW;
25 Feb 2002: Urszula Jambor; check on both SW & LW file status before
             using
04 Jun 2002: Urszula Jambor; allowed fall back to model SW.
10 Dec 2002: Urszula Jambor; replaced lis%astat1,2 with local sstat1,2
             Reorganized routine to mirror other get-routines, and
             corrected bug in file status check for initial time1.
```

INTERFACE:

```
subroutine getgrad
```

USES:

```
use lisdrv_module, only : lis, grid
use obsradforcing_module, only : obswdata1,obswdata2,oblwdata1,oblwdata2,&
    sstat1,sstat2,lstat1,lstat2
use time_manager
use time_module
use agrmetdomain_module, only : agrmetdrv

implicit none
```

CONTENTS:

```
!-----
! Determine Required Observed Radiation Data Times
!-----
yr1 = lis%t%yr      !Previous Hour
mo1 = lis%t%mo
da1 = lis%t%da
hr1 = lis%t%hr
mn1 = 0
ss1 = 0
ts1 = 0
call tick ( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

yr2 = lis%t%yr      !Next Hour
mo2 = lis%t%mo
da2 = lis%t%da
hr2 = lis%t%hr
mn2 = 0
ss2 = 0
ts2 = 60*60
call tick ( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )

lis%f%findagrtime1=0
lis%f%findagrtime2=0
movetime=0

if(lis%t%time.ge.agrmetdrv%agrmtime2) then
    movetime=1
    lis%f%findagrtime2=1
endif

if(get_nstep()==0 .or.get_nstep()==1) then
    lis%f%findagrtime1=1
    lis%f%findagrtime2=1
    movetime=0
```

```

        endif

        if (lis%f%findagrtme1==1) then
            call agrSWfile ( nameSH, lis, yr1, mo1, da1, hr1 )
            print*, 'calling swfile.. ',nameSH
            sstat1 = 0
            call retglbSW ( 1, nameSH, sstat1, 1 )
            lstat1 = 0
            call retagrlw ( 1, yr1, mo1, da1, hr1, lstat1, 1 )
            if ((sstat1 + lstat1) < 2) then
                sstat1 = 0
                lstat1 = 0
            end if
            if (sstat1 /= 0) agrmetdrv%agrmtme1 = time1
        endif
        if(movetime.eq.1) then
            agrmetdrv%agrmtme1 = agrmetdrv%agrmtme2
            sstat1 = sstat2
            lstat1 = lstat2
            do c=1, lis%d%ngrid
                obswdata1(c) = obswdata2(c)
                oblwdta1(c) = oblwdta2(c)
            end do
        endif

        if(lis%f%findagrtme2.eq.1) then
            call agrSWfile ( nameSH, lis, yr2, mo2, da2, hr2 )
            sstat2 = 0
            call retglbSW ( 2, nameSH, sstat2, 1 )
            lstat2 = 0
            call retagrlw ( 2, yr2, mo2, da2, hr2, lstat2, 1 )
            if ((sstat2 + lstat2) < 2) then
                sstat2 = 0
                lstat2 = 0
            end if
            if (sstat2 /= 0) agrmetdrv%agrmtme2 = time2
        endif
!-----
! Print out Status of data holdings
!-----
        if (lis%t%time == time1) then
            if (sstat1==0) write(79,*) 'NO AGR SW USED',mo1,da1,yr1,hr1
            if (sstat1/=0) write(79,*) 'USED AGRMET SW',mo1,da1,yr1,hr1
            if (sstat2==0) write(79,*) 'NO AGR SW USED',mo2,da2,yr2,hr2
            if (sstat2/=0) write(79,*) 'USED AGRMET SW',mo2,da2,yr2,hr2

            if (lstat1==0) write(79,*) 'NO AGR LW USED',mo1,da1,yr1,hr1
            if (lstat1/=0) write(79,*) 'USED AGRMET LW',mo1,da1,yr1,hr1

```

```

if (lstat2==0) write(79,*) 'NO AGR LW USED',mo2,da2,yr2,hr2
if (lstat2/=0) write(79,*) 'USED AGRMET LW',mo2,da2,yr2,hr2
endif
return

```

1.49.2 agrSWfile: (Source File: getgrad.F90)

This subroutine puts together the radiation data filenames.

REVISION HISTORY:

```

28 Oct 1999: Brian Cosgrove; Initial code
18 Jun 2001: Urszula Jambor; Modified for AGRMET data use in GLDAS
24 Oct 2001: Jesse Meng; Modified for AGRMET directory structure
15 Aug 2003: Sujay Kumar: Modified to create a global filename
                  instead of two filenames for each hemisphere.

```

INTERFACE:

```
subroutine agrSWfile ( nameSH, lis, yr, mo, da, hr )
```

USES:

```

use lis_module
use agrmetdomain_module, only : agrmetdrv

```

CONTENTS:

```

92 format (80a1)
93 format (a80)
94 format (a5, i4, i2, i2, i2)
95 format (15a1)
96 format (a40)
97 format (a1)
98 format (a6, i4, i2, a1)
99 format (13a1)

open(unit=81, file='temp', form='formatted', access='direct', recl=80)
write(81,96,rec=1) agrmetdrv%agrmetdir
read(81,92,rec=1) (fbase(i), i=1,80)

write(81,98,REC=1) '/SWDN/' ,yr,mo,'/'
read(81,99,rec=1) fdir
do i = 1, 13
  if ( fdir(i) == ' ') fdir(i) = '0'
end do
write(81,94,rec=1) 'swdn_', yr, mo, da, hr
read(81,95,rec=1) ftime

```

```

do i = 1, 15
    if ( ftime(i) == ' ') ftime(i) = '0'
end do
c = 0
do i = 1, 80
    if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(81, 92, rec=1) (fbase(i),i=1,c), (fdir(i),i=1,13), &
(ftime(i),i=1,15)
read(81, 93, rec=1) nameSH
close(81)

return

```

1.49.3 subroutine interp_agrmet_lw.F90 (Source File: interp_agrmet_lw.F90)

Opens, reads, and interpolates AGRMET longwave radiation
GRIB IPOLATES UTILITY TO INTERPOLATE DATA FOR NH AND SH IN RT-
NEPH (512X512) POLAR STEREOGRAPHIC GRIDS TO MERGED GLOBAL DATA
IN GLDAS-SPECIFIED LAT/LON GRIDS;

REVISION HISTORY:

26 Jun 2001: Urszula Jambor; Initial code, based on Jesse Meng's
RTNEPH2LATLON.F code.
08 Feb 2002: Urszula Jambor; Modified declarations of arrays
dependant on domain & resolution to allocatable.
Pass in values for latmax.
11 Dec 2002: Urszula Jambor; Added 1/2 & 1 degree resolution GDS arrays

INTERFACE:

```
subroutine interp_agrmet_lw( pdata1, outdata, ferror )
```

USES:

```

use lisdrv_module,only : lis,gindex
use agrmetdomain_module, only : rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22, mi
implicit none

integer, parameter :: nagrc = 1440, nagrr=600

```

ARGUMENTS:

```

real :: pdata1(nagrc, nagrr)
real :: outdata(lis%d%ngrid) !output array matching grid(c,r)
integer :: ferror           !set to zero if error found

```

CONTENTS:

```

!-----
! READ INPUT DATA
!-----
allocate(pdata(mi))
allocate(lo1(lis%d%lnc*lis%d%lnr))
allocate(ldata1(lis%d%lnc*lis%d%lnr))
if (ferror == 0) then
    do i=1,lis%d%ngrid
        outdata(i) = lis%d%undef
    end do
else
    ferror = 1
    ibi = 1
    count = 0
    li1 = .false.
    do j=1,nagrr
        do i=1,nagrc
            pdata(count+i) = pdata1(i,j)
        enddo
        count = count+nagrc
    enddo
    do i=1,mi
        if(pdata(i).eq.-9999) then
            li1(i) = .false.
        else
            li1(i) = .true.
        endif
    enddo
    kgdso = 0
    kgdso = lis%d%kgds
    call polates0(kgdso,ibi,li1,pdata,ibo,lo1,lidata1,mi,&
                  rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)

    if(iret .NE. 0) then
        print*, "IPOLATES ERROR!! PROGRAM STOP!!"
        call exit(iret)
    end if
!-----
! COMBINE LDATA1 AND LDATA2 INTO LDATA
!-----
count = 0
do j=1,lis%d%lnr
    do i=1,lis%d%lnc
        if(gindex(i,j).ne. -1) then
            outdata(gindex(i,j)) = ldata1(i+count)
        endif
    enddo
end do

```

```

    enddo
    count = count+lis%d%lnc
  enddo
endif
deallocate(pdata)
deallocate(lo1)
deallocate(ldata1)

```

1.49.4 interp_agrmet_sw.F90 (Source File: interp_agrmet_sw.F90)

Opens, reads, and interpolates AGRMET shortwave radiation forcing

REVISION HISTORY:

26 Jun 2001: Urszula Jambor; Initial code, based on Jesse Meng's RTNEPH2LATLON.F code.
 08 Feb 2002: Urszula Jambor; Modified declarations of arrays dependant on domain & resolution to allocatable.
 Pass in values for latmax.
 11 Dec 2002: Urszula Jambor; Added 1/2 & 1 degree resolution GDS arrays

INTERFACE:

```
subroutine interp_agrmet_sw( nameSH, outdata, ferror )
```

USES:

```
use lisdrv_module,only : lis,gindex
use agrmetdomain_module, only : rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22, mi
implicit none
```

ARGUMENTS:

```
character*80 :: nameSH
real :: outdata(lis%d%ngrid)
integer :: ferror
```

CONTENTS:

```

allocate(pdata(mi))
allocate(lo1(lis%d%lnc*lis%d%lnr))
allocate(ldata1(lis%d%lnc*lis%d%lnr))
print*, 'Reading AGRMET file : ',nameSH
open(11, file=nameSH, form="unformatted", iostat=openerrN)
read(11, iostat=readerrN) pdata1
close(11)

if ((openerrN+openerrS+readerrN+readerrS) > 0) then

```

```

ferror = 0
if ((openerrS+readerrS) > 0) then
    print*, 'AGRMET file problem:', nameSH
end if
do i=1,lis%d%ngrid
    outdata(i) = lis%d%undef
end do
else
    ferror = 1
    ibi = 1
    count = 0
    li1 = .false.
    do j=1,nagrr
        do i=1,nagrc
            pdata(count+i) = pdata1(i,j)
        enddo
        count = count+nagrc
    enddo
    do i=1,mi
        if(pdata(i).eq.-9999) then
            li1(i) = .false.
        else
            li1(i) = .true.
        endif
    enddo
    kgdso = 0
    kgdso = lis%d%kgds
    call polates0(kgdso,ibi,li1,pdata,ibo,lo1,lidata1,mi,&
                  rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)

    if(iret .NE. 0) then
        print*, "IPOLATES ERROR!! PROGRAM STOP!!"
        call exit(iret)
    end if
    count = 0
    do j=1,lis%d%lnr
        do i=1,lis%d%lnc
            if(gindex(i,j).ne. -1) then
                outdata(gindex(i,j)) = ldata1(i+count)
            endif
        enddo
        count = count+lis%d%lnc
    enddo
endif
deallocate(pdata)
deallocate(lo1)
deallocate(lidata1)

```

1.49.5 readagrmecrd.F90 (Source File: readagrmecrd.F90)

Routine to read AGRMET specific parameters from the card file.

REVISION HISTORY:

11 Dec 2003; Sujay Kumar, Initial Code

INTERFACE:

```
subroutine readagrmecrd(agrmecrdv)
```

USES:

```
use agrmetdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=agrmec)
print*, 'Using AGRMET forcing'
print*, 'AGRMET forcing directory :', agrmetdrv%AGRMETDIR
agrmecrv%AGRMTIME1 = 3000.0
agrmecrv%AGRMTIME2 = 0.0

close(11)
```

1.49.6 retagrlw.F90 (Source File: retagrlw.F90)

Opens, reads, interpolates and overlays LW radiation forcing.

TIME1 = most recent past data

TIME2 = most recent future data

1. Load AGRMET 3 level cloud amount (converted from RTNEPH)
2. Convert 2 m forcing specific humidity to vapor pressure
3. Use AGRMET subroutine longwv() to calculate LW DOWN. Arguments are:
cloud amount [%], 2 m temperature [K], and 2 m vapor pressure [Pa]

REVISION HISTORY:

```
28 Oct 1999: Brian Cosgrove; Initial code
11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With inland
              water filled in). Deleted unused variables.
20 Jun 2000: Brian Cosgrove; changed code so that it uses LDAS%UDEF and
              not a hard-wired undefined value of -999.9 and -999.0
25 Jun 2001: Urszula Jambor; Modified for AGRMET data use in GLDAS.
25 Oct 2001: Jesse Meng; Modified for AGRMET LW scheme implementation
```

INTERFACE:

```
subroutine retagrlw ( order,yr, mo, da, hr, ferror, flag )
```

USES:

```
use lisdrv_module, only : lis, grid
use obsradforcing_module, only : oblodata1,oblodata2
use agrmetdomain_module, only : agrmetdrv
implicit none
```

ARGUMENTS:

```
integer :: yr, mo, da, hr
integer :: order           !retrieve data for time1 or time2
integer :: ferror          !0=no radiation data found
                           !1=found observational data
```

CONTENTS:

```
!-----
! If using AGRMET data, open, read in and interpolate AGRMET files
! to appropriate GLDAS resolution;
! If error reading file, or data completely missing, ferror=0;
!-----
ferror = 0
do c = 1, lis%ngrid
  if (order == 1) then
    oblodata1(c) = lis%d%undef
  else if (order == 2) then
    oblodata2(c) = lis%d%undef
  end if
end do

if (flag == 1) then
!-----
! 1. Generate AGRMET cloud filenames
! Load 3 layers cldamt data
!-----

      write(cyr, '(I4.4)') yr
      write(cmo, '(I2.2)') mo
      write(cda, '(I2.2)') da
      write(chr, '(I2.2)') hr

      nameSH = trim(agrmtdrv%agrmetdir)//'CloudAGR'//cyr//cmo// &
                 '/cldamt_ '//cyr//cmo//cda//chr
      print*, 'Reading AGRMET file : ',nameSH
      open(11, file=nameSH, status='old',access='direct',&
            recl=3456000, iostat=openerrN)
      read(11, rec=1,iostat=readerrN) pdata1H
```

```

        read(11, rec=2,iostat=readerrN) pdata1M
        read(11, rec=3,iostat=readerrN) pdata1L
        close(11)
        if ((openerrN+readerrN) > 0) then
            ferror = 0
            print*, 'AGRMET file problem: ',nameSH
        else
            ferror = 1
        endif
        call interp_agrmel_lw( pdata1H,cldamtH, ferror )
        call interp_agrmel_lw( pdata1M,cldamtM, ferror )
        call interp_agrmel_lw(pdata1L,cldamtL, ferror )
        if ( ferror .EQ. 0 ) return
!-----
! If AGRMET cloud data is not undefined, set ferror=1
!-----
        fvalid = 0
        do c=1, lis%d%ngrid
            if (cldamtH(c) >= 0.0) fvalid = 1
            if (cldamtM(c) >= 0.0) fvalid = 1
            if (cldamtL(c) >= 0.0) fvalid = 1
        end do
        if ( fvalid .EQ. 0 ) return
        do c=1,lis%d%ngrid
            rldown = 0.
            tair = grid(c)%forcing(1)
!-----
! 2. CONVERT 2 METER SPECIFIC HUMIDITY TO VAPOR PRESSURE
!-----
            vaporP = grid(c)%forcing(2) * grid(c)%forcing(7) / &
                ( 0.622 + grid(c)%forcing(2) * (1-0.622) )
!-----
! If tair, vaporP, and ALL 3 AGRMET LAYERS cldamt are defined,
! calculate rldown; transfer to proper output array
!-----
            fvalid = 1
            if (tair.LT. 0.0) fvalid = 0
            if (vaporP.LT. 0.0) fvalid = 0
                if (cldamtH(c) .LT. 0.0) fvalid = 0
                if (cldamtM(c) .LT. 0.0) fvalid = 0
                if (cldamtL(c) .LT. 0.0) fvalid = 0

            if (fvalid == 1) then
                cldamt1d(1) = cldamtL(c)
                cldamt1d(2) = cldamtM(c)
                cldamt1d(3) = cldamtH(c)
!-----
! 3. Calculate lw down

```

```

!-----
      call agrlwdn( tair, vaporP, cldamt1d, rldown )
      if (order==1) then
          OBLWDATA1(c) = rldown
      else if (order==1) then
          OBLWDATA2(c) = rldown
      end if
      else
          if (order==1) then
              OBLWDATA1(c) = lis%d%udef
          else if (order==1) then
              OBLWDATA2(c) = lis%d%udef
          end if
      end if
      end do
  end if
  return

```

1.49.7 retglbSW.F90 (Source File: retglbSW.F90)

Opens, reads, and interpolates radiation forcing.

TIME1 = most recent past data

TIME2 = most recent future data

REVISION HISTORY:

```

28 Oct 1999: Brian Cosgrove; Initial code, retrad.f
25 Jun 2001: Urszula Jambor; Renamed & modified for AGRMET data
               use in GLDAS.
21 Nov 2001: Urszula Jambor; Added iostat check in open & read stmts;
               renamed to distinguish SW from LW routines.
27 Feb 2002: Urszula Jambor; Added call to fill_land to compensate
               deficiencies in interpolation to coarse 2x2.5 grid.
16 Oct 2002: Urszula Jambor; Corrected array ranges used in 2x2.5 case.
               Previously, array mismatch occurred.
11 Dec 2002: Urszula Jambor; Added domain 4 and 5 in attached routine

```

INTERFACE:

```
subroutine retglbSW ( order, nameSH, ferror, flag )
```

USES:

```

use lisdrv_module,only : lis, grid           ! LDAS non-model-specific 1-D variables
use obsradforcing_module, only : obswdata1, obswdata2
implicit none

```

ARGUMENTS:

```

character*80 :: nameSH
integer :: flag           !data source, 1=AGRMET
integer :: order          !retrieve data for time1 or time2
integer :: ferror         !0=no radiation data found
                           !1=found observational data (may be undef)

```

CONTENTS:

```

!-----
! If using AGRMET data, open, read in and interpolate AGRMET files
! to appropriate GLDAS resolution
! If error reading file, ferror=0 else ferror=1
!-----
if (flag == 1) then
  call interp_agrmet_sw( nameSH, outdata, ferror )
  do c=1, lis%d%ngrid
    if (order == 1) then
      obswdata1(c) = outdata(c)
    else if (order == 2) then
      obswdata2(c) = outdata(c)
    end if
  end do
end if !flag=1

```

1.50 Fortran: Module Interface time_interp_agrmet.F90 (Source File: time_interp_agrmet.F90)

This routine consists of utilities for temporal interpolation of radiation forcing
 TIME1 = most recent past data TIME2 = most recent future data

REVISION HISTORY:

```

28 Oct 1999: Brian Cosgrove; Initial code, see getrad.f
27 Apr 2000: Brian Cosgrove' Disabled zenith angle correction cutoff for
             cos(zen) less than .2
11 May 2000: Brian Cosgrove; Enabled correction cutoffs for cos(zen) less
             than .1, stop model if computed value is greater than 1367 w/m2
08 Jan 2001: Brian Cosgrove; Added check to see if czb or czm is equal
             to zero before trying to divide by czm or czb.  If it is
             zero, set radiation value to zero
06 Mar 2001: Brian Cosgrove; Changed computation of WT1 and WT2 in cases
             where the previous hour or the next hour of observed radiation
             is not available. Substituted TIME1 for LDAS%PINKTIME1 and
             LDAS%NESTIME1 and TIME2 for LDAS%PINKTIME2 and LDAS%NESTIME2
15 Jun 2001: Urszula Jambor; Reworked algorithm for AGRMET data & GLDAS.
15 Oct 2001: Jesse Meng; Replace lis%agrmet flag by lis%agrmetsw and

```

```

        lis%agrmetlw; added call retagrlw() to calculate
        AGRMET LW;
25 Feb 2002: Urszula Jambor; check on both SW & LW file status before
               using
04 Jun 2002: Urszula Jambor; allowed fall back to model SW.
10 Dec 2002: Urszula Jambor; replaced lis%astat1,2 with local sstat1,2
               Reorganized routine to mirror other get-routines, and
               corrected bug in file status check for initial time1.

```

INTERFACE:

```
subroutine time_interp_agrmet
```

USES:

```

use lisdrv_module, only : lis, grid
use obsradforcing_module, only : obswdata1, obswdata2,oblwdata1,oblwdata2,&
      sstat1,sstat2,lstat1,lstat2
use grid_spmdMod
use time_module
use agrmetdomain_module, only : agrmetdrv
implicit none

```

CONTENTS:

```

!-----
! Loop through and replace data as possible with AGRMET data
! This depends on options specified in lis.crd as well as actual
! data holdings.
!-----
allocate(obsw(gdi(iam)))
yr1 = lis%t%yr      !Previous Hour
mo1 = lis%t%mo
da1 = lis%t%da
hr1 = lis%t%hr
mn1 = 0
ss1 = 0
ts1 = 0
call tick ( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

yr2 = lis%t%yr      !Next Hour
mo2 = lis%t%mo
da2 = lis%t%da
hr2 = lis%t%hr
mn2 = 0
ss2 = 0
ts2 = 60*60
call tick ( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
#if(defined SPMD)
call MPI_BCAST(agrmtdrv%agrmtime1,1,MPI_REAL8,0,&
```

```

MPI_COMM_WORLD,iер)
call MPI_BCAST(agrmtdrv%agrmtim2,1,MPI_REAL8,0,&
    MPI_COMM_WORLD,iер)
call MPI_BCAST(lis%t%time,1,MPI_REAL8,0,&
    MPI_COMM_WORLD,iер)
call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0,&
    MPI_COMM_WORLD,iер)
#endif
if ((sstat1 == 1) .and. (sstat2 == 1)) then
!-----
! Compute weights and zenith angle information. Replace forcing
! with AGRMET data.
!-----
wt1 = (agrmtdrv%agrmtim2 - lis%t%time) / (agrmtdrv%agrmtim2 - agrmtdrv%agrmtim1)
wt2 = 1.0 - wt1
do c=1,gdi(iam)
    zdoy = lis%t%doy
    call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
        lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
    osw(c) = lis%d%udef
    if ((obswdata1(c)>0.0) .and. (obswdata2(c)>0.0)) then
        obsw(c) = obswdata1(c)*zw1+obswdata2(c)*zw2
        if ((obsw(c) > obswdata1(c) .and. &
            obsw(c) > obswdata2(c)) .and. &
            (czb<0.1 .or. cze<0.1)) then
            obsw(c) = obswdata1(c)*wt1+obswdata2(c)*wt2
        end if
    end if
    if ((obswdata1(c) > 0.0) .and. &
        (obswdata2(c) <= 0.0) ) then
        if (czb > 0.0) then
            obsw(c) = obswdata1(c) * (czm/czb)
        else
            obsw(c) = obswdata1(c) * (0.0)
        end if
        if ((obsw(c) > obswdata1(c) .and. &
            obsw(c) > 0.0) .and. (czb<0.1 .or. cze<0.1)) then
            obsw(c) = obswdata1(c)*wt1 + 0.0*wt2
        end if
    end if
    if ((obswdata1(c)<=0.0) .and. (obswdata2(c)>0.0)) then
        if (cze > 0.0) then
            obsw(c) = obswdata2(c) * (czm/cze)
        else
            obsw(c) = obswdata2(c) * (0.0)
        end if
        if ((obsw(c)>0.0 .and. obsw(c)> obswdata2(c)) &
            .and. (czb < 0.1 .or. cze < 0.1)) then

```

```

        obsw(c) = 0.0*wt1 + obswdata2(c)*wt2
    end if
end if

if (obsw(c) >= 0.0) then
    if (obsw(c) .gt. 1367) then
        print *, 'FALLING BACK TO MODEL SW'
    else
        grid(c)%forcing(3) = obsw(c)
    end if
end if
end do !c
end if

if ((sstat1 == 1) .and. (sstat2 /= 1)) then
!---
! Compute weights and zenith angle information. Replace forcing
! with zenith extrapolated AGRMET data
!---
wt1 = (time2 - lis%t%time) / (time2 - agrmetdrv%agrmttime1)
wt2 = 1.0 - wt1
do c=1, gdi(iam)
    zdoy = lis%t%doy
    call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
                lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
    obsw(c) = lis%d%udef
    if (obswdata1(c) > 0.0) then
        if (czb > 0.0) then
            obsw(c) = obswdata1(c) * (czm/czb)
        else
            obsw(c) = obswdata1(c) * (0.0)
        end if
        if ((obsw(c) > 400.0) .and. &
            (czb < 0.1 .or. cze < 0.1) ) then
            obsw(c) = obswdata1(c)*wt1
        end if

        if (obsw(c) >= 0.0) then
            if (obsw(c) .gt. 1367) then
                print *, 'warning, OBSERVED SW RADIATION HIGH'
                print *, 'it is',grid(c)%forcing(3), 'at',c
                print *, 'agrm1=',obswdata1(c)
                print *, 'agrm2=',obswdata2(c)
                print *, 'wt1,wt2,czb,cze,czm,zw1,zw2'
                print *,wt1,wt2,czb,cze,czm,zw1,zw2
                print *, 'FALLING BACK TO MODEL SW'
            else
                grid(c)%forcing(3) = obsw(c)
            end if
        end if
    end if
end do

```

```

        end if
    end if
end if
end do !c
end if

if ((sstat1 /= 1) .and. (sstat2 == 1)) then
!---
! Compute weights and zenith angle information. Replace forcing
! with zenith extrapolated AGRMET data
!---
wt1 = (agrmetdrv%agrmtime2 - lis%t%time) / (agrmetdrv%agrmtime2 - time1)
wt2 = 1.0 - wt1
do c=1, gdi(iam)
    zdoy = lis%t%doy
    call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
        lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
    obsw(c) = lis%d%udef
    if (obswdata2(c) > 0.0) then
        if (cze > 0.0) then
            oobs(c) = oobsdata2(c) * (czm/cze)
        else
            oobs(c) = oobsdata2(c) * (0.0)
        end if
        if ((obsw(c) > 400.0) .and. &
            (czb < 0.1 .or. cze < 0.1) ) then
            oobs(c) = 0.0*wt1 + oobsdata2(c)*wt2
        end if

        if (obsw(c) >= 0.0) then
            if (obsw(c) .gt. 1367) then
                else
                    grid(c)%forcing(3) = oobs(c)
                end if
            end if
        end if
    end do !c
end if
if ((sstat1 /= 1) .and. (sstat2 /= 1)) then
    do c=1, gdi(iam)
        oobs(c) = lis%d%udef
    end do
end if
if ((lstat1 == 1) .and. (lstat2 == 1)) then
    wt1 = (agrmetdrv%agrmtime2 - lis%t%time) / &
        (agrmetdrv%agrmtime2 - agrmetdrv%agrmtime1)
    wt2 = 1.0 - wt1
    do c=1,gdi(iam)

```

```
if ( (OBLWDATA1(c) > 0.0) .AND. &
      (OBLWDATA2(c) > 0.0) ) then
  grid(c)%forcing(4)= oblwdatal(c)*wt1 &
  + oblwdat2(c)*wt2
end if
end do
end if
deallocate(obsw)
return
```